JTC 1

TECHNICAL REPORT

ISO/IEC TR 9572

First edition
1989-09-15

Information technology — Open Systems Interconnection — LOTOS description of the session protocol

Traitement de l'information — Interconnexion de systèmes ouverts — Description en LOTOS du protocole de session

Reference number
ISO/IEC/TR 9572 : 1989 (E)

Contents

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) together form a system for worldwide standardization as a whole. 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 a technical committee is to prepare International Standards but in exceptional circumstances, the publication of a technical report of one of the following types may be proposed:

- type 1, when the necessary support within the technical committee cannot be obtained for the publication of an International Standard, despite repeated efforts;

- type 2, when the subject is still under technical development requiring wider exposure;

- 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 Technical reports of type 3 do not necessarily have to be reviewed until the data they provide are considered to be no longer valid or useful.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 9572, which is a technical report of type 2, was prepared by ISO/IEC JTC 1, *Information technology*.

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 important approach for providing such definitions. The use of Formal Description Techniques in this area is however relatively new and their application on a wide scale cannot be expected overnight. Formal descriptions should be introduced gradually in standards if initially the number of Member Bodies that are able to contribute to their development is too small, thus allowing time to gain experience and to develop educational material.

An ad-hoc group for the formal description of the Session Layer, i.e. of the Session Service ISO 8326 and the Session Protocol ISO 8327, was established in November 1985. This group applied the Formal Description Technique LOTOS, defined in ISO 8807, which at that time was still under development. In September 1986 two Working Documents were produced which contained the LOTOS draft specifications of ISO 8326 and ISO 8327 respectively. As a byproduct, the group also produced a number of Defect Reports on the standards, most of which have been accepted and incorporated in the standards.

A Ballot was then issued requesting Member Bodies to state their position concerning the progression of the formal descriptions. Based on this Ballot, SC21 decided in June 1987 to progress both formal descriptions as Type 2 Technical Reports. The main reason for not incorporating them into the standards was that Member Bodies expressed their current lack of expertise on the subject. It seemed therefore appropriate that a period of time passed, during which the formal decriptions can be read and compared with the standards, and after which the status and progression of the formal descriptions can be re-evaluated.

The purpose of this Technical Report is to provide a complete, consistent and unambiguous description of ISO 8327. It forms therefore a companion document to ISO 8327. It takes account of the Defect Reports incorporated in the standard (annex D of ISO 8327), however, it does not necessarily take account of subsequent amendments or addenda to the standard.

Information technology - Open Systems Interconnection - LOTOS description of the session protocol

1 Scope

This Technical Report contains a formal description of the OSI Basic Connection Oriented Session Protocol defined in ISO 8327. The formal definitions presented in this Technical Report are expressed in the formal description technique LOTOS, which is defined in ISO 8807.

These formal definitions are related to the formal descriptions in LOTOS of the OSI Connection Oriented Session Service, ISO 8326, and of the OSI Transport Service, ISO 8072. Moreover, formal definitions of these services, contained in ISO/IEC/TR 9571 and ISO/IEC/TR 10023, respectively, are used and referenced in this Technical Report.

The formal description is not limited to a single session protocol machine, but also describes multiple session protocol machines that result from support of multiple session connections either in parallel or in sequence. Therefore, it also formalizes aspects of multiplicity which are not presented in ISO 8327 directly, but by way of reference to the OSI Basic Reference Model, ISO 7498.

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 agreement based on this Technical Report are encouraged to investigate the possibility of applying the most recent editions of the standards listed below. Members of ISO and IEC maintain registers of 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 8326: 1987, *Information processing systems - Open Systems Interconnection - Basic connection oriented session service definition.*

ISO 8327: 1987, *Information processing systems - Open Systems Interconnection - Basic connection oriented session protocol specification.*

ISO 8807: 1988, *Information processing systems - Open Systems Interconnection - LOTOS - A formal description technique based on the temporal ordering of observational behaviour.*

ISO/IEC/TR 9571: 1989, *Information processing systems - Open Systems Interconnection - LOTOS description of the session service.*

ISO/IEC/TR 10023: - [1], *Information processing systems - Open Systems Interconnection - LOTOS description of the transport service.*

3 Definitions

For the purpose of this Technical Report the definitions given in ISO 8327 apply.

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, and uses the abbreviations contained in clause 4 of ISO 8327.

The following additional abbreviations are employed in this Techical Report:

SC	session connection
SCEP	session connection endpoint
SCEI	session connection endpoint identifier
SSP	session service primitive
TC	transport connection
TS	Transport Service
TSDU	transport service data unit

5 Conventions

Clauses 6 through 10 of this Technical Report constitute LOTOS text. All informal explanations in these clauses form LOTOS comments. They are thus separated from the LOTOS specifications (of data types and dynamic behaviour) according to the rules for comment delimitation. Moreover, informal explanations precede the formal definitions to which they refer and contain a final line of only "-" characters. Informal explanations following formal definitions contain a first line of only "-" characters.

Formal definitions, as well as formal symbols and identifiers referenced in informal explanations, are printed in italics.

1) To be published.

(* start of LOTOS text ----------

6 Introduction to the formal description

The formal description relates to the dynamic behaviour of an unbounded number of SPMs, each of which supports the provision of a single SC or, in case of re-use of the underlying TC, a sequence of SCs. The SS boundary is formally represented by a single gate *s* and the TS boundary by a single gate *t*. Events at these gates are structured as defined in the service formal descriptions, ISO/IEC/TR 9571 and ISO/IEC/TR 10023. Constraints on connection identification, acceptance of connections and backpressure flowcontrol (from the TS-provider) apply to the Session Protocol, as well as to the related services, and the processes representing these constraints are in fact imported from the service formal descriptions. Figure 1 shows the conjunction of processes resulting from this top level decomposition.

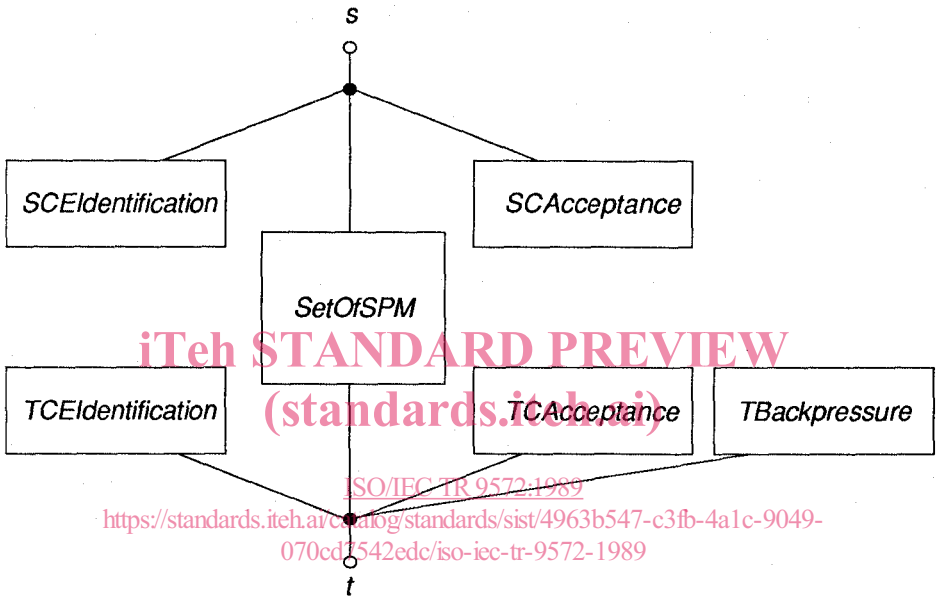

Figure 1 - (Top level) processes representing separate constraints for a Session Protocol entity

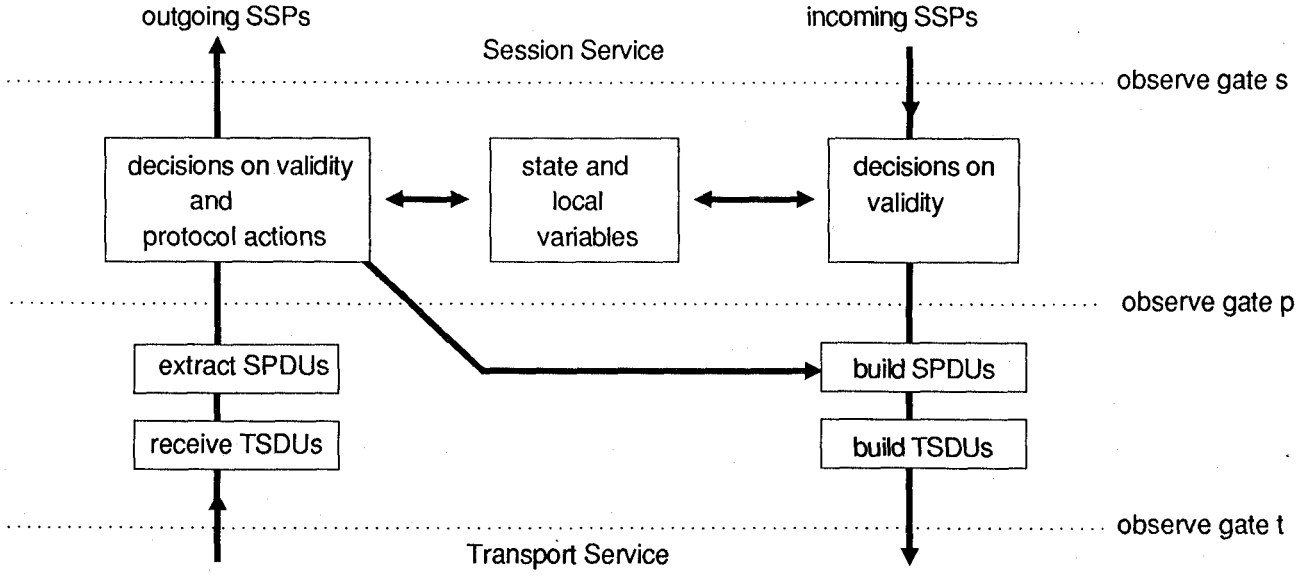

Figure 2 - Actions within a SPM

An internal gate *p* is introduced to favour separation of concerns based on the concept of SPDU. SPDUs are transferred via the TS by encoding, and possibly concatenating, them into TS data units. The interactions at gate *p* are independent of encoding and concatenation, however, thus only abstract SPDUs are to be considered at *p*. The actions of a single SPM are summarized in figure 2.

The behaviour of the SPM is described by specifying the constraints on the behaviour observed at the three gates *s*, *p* and *t*. This is obtained by a parallel composition of a process, termed *STPM*, that roughly corresponds to the state tables protocol machine, together with processes that enforce the local constraints on service primitives at connection endpoints. Again, the latter constraints can be specified with instances of processes imported from the SS and TS formal descriptions (namely *SCEP* and *TCEP*).

The next level of substructuring is found in the decomposition of *STPM* into processes representing the following separate constraints:

a) The relationship between SSPs and TSPs that does not involve abstract SPDUs (described by process *SSPTSP*). This constraint is concerned with "direct mappings" between service primitives, where no SPM generated information is needed to coordinate the interworking with the remote SPM. Also the establishment of a TC is attributed to this constraint.

b) The relationship between SSPs and abstract SPDUs, including segmentation of SSDUs (described by process *SSPSPDU*).

c) The transformation of abstract SPDUs to TSPs, and vice versa, including encoding and concatenation of SPDUs (described by process *SPDUTSP*).

d) The constraints on the ordering and contents of abstract SPDUs, corresponding to the state tables description in annex A of ISO 8327 (described by process *SPDUConstraints*).

SSPTSP and *SSPSPDU* synchronize at gate *s*, *SSPTSP* and *SPDUTSP* synchronize at gate *t*, and, finally, *SSPSPDU*, *SPDUTSP* and *SPDUConstraints* synchronize at gate *p*. This is shown in figure 3. The style of the formal description, which now also shows a limited internal structure, can be characterized as a mixture of constraint- and resource-oriented styles.

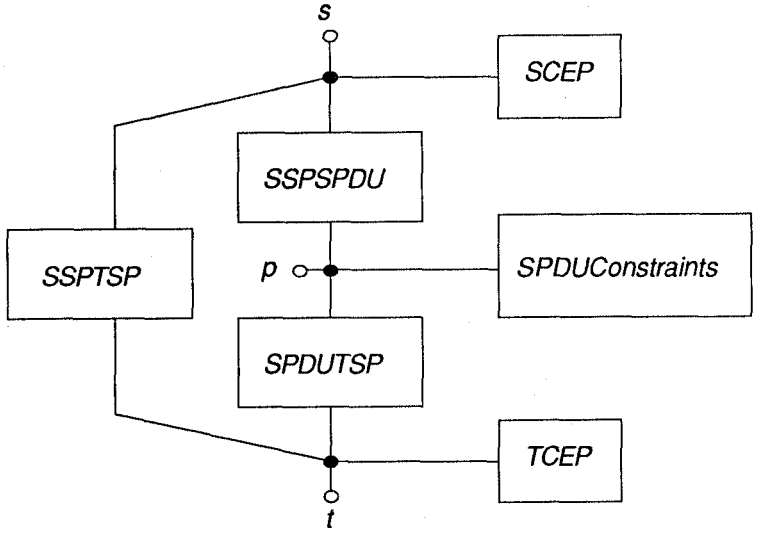

Figure 3 - Conjunction of processes representing the behaviour of a single SPM

Each of the processes that must synchronize at *p* imposes its own constraints on the exchange of abstract SPDUs. Because of this, an abstract SPDU exchange only occurs if all processes agree on that particular

exchange. The event structure of the abstract SPDU exchange is as follows:

p ?pd:ASPDU ?fl:TFlow ?d:Dir ?v:Validity ?st:TrState

with components

- *pd* of sort *ASPDU*: the abstract SPDU being exchanged;
- *fl* of sort *TFlow*: the transport flow, which is either normal or expedited;
- *d* of sort *Dir*: the direction of transfer, which is either sending or receiving;
- *v* of sort *Validity*: the "status" of the abstract SPDU (only applicable for received SPDUs). According to the state tables of ISO 8327, processing of a received SPDU, and in particular determining its validity, depends upon the result of several actions (see figure 2) and/or the values of certain session variables. Relevant states that involve more than one process can be established through this component when synchronizing at *p*; and
- *st* of sort *TrState*: the "coarse-grained" state of the SPM (only applicable for received SPDUs and for sending the DN SPDU). The SPM state, known to process *SPDUConstraints*, is often also required by other processes upon exchange of a SPDU. However, rather than the fine-grained division of the data transfer phase into SPM states, certain collections of SPM states are relevant for the latter processes. Values of this component are used to represent such collections and can be communicated when synchronizing at *p*.

NOTES

1 - In order to represent flow control and segmenting, events at gates *s* and *t* may represent the exchange of single data octets instead of just executions of "complete" service primitives. Octet-wise exchange of data is only modelled for those service primitives whose user data parameter has no length restrictions (i.e., S-DATA, S-TYPED-DATA and T-DATA). Except for the need of "data octet" events, the characteristics of octet-wise exchange of data are

- two events, a start and an end event, are significant to delimitate an exchange of data;
- the state transition and predicates that are associated with a SSP request are now associated with the corresponding start event. If a colliding or overtaking, that is, "intervenient", event occurs before the end event, then either
  a) the (unfinished) request is not disrupted (provided the intervenient event is not destructive), but will be resumed after the intervenient event, or
  b) the (unfinished) request is disrupted, and the corresponding TSP request is cancelled or terminated;
- the state transition and predicates that are associated with a TSP indication are now associated with the corresponding end event. If a destructive or overtaking event occurs before the end event, then the (unfinished) indication is disrupted, and all previous events of the indication are discarded.

2 - The formal description extends beyond the state tables description of ISO 8327 by way of explicitly describing

- the constraints related to the multiplicity of SPMs;
- concatenation, extended concatenation and segmentation;
- the transfer of expedited data;
- the handling of invalid and unrecognizable events.

Figure 4 gives an overview of the processes used for the formal definition of the Session Protocol and their relations (a number of the lowest level processes are omitted). It also indicates the clauses where the definition of these processes can be found ("SS" means: defined in ISO/IEC/TR 9571; "TS" means: defined in ISO/IEC/TR 10023).

The definition of data types precedes the definition of the dynamic behaviour in which these types are used. A number of standard data types are imported from the LOTOS library of data types. Apart from these, and some "ad-hoc" data types, the types can roughly be divided in types for the representation of abstract SPDUs, of encoded SPDUs, and of session variables. Additionally, a number of types are related to the components of an internal event at gate *p* (besides the abstract SPDU).

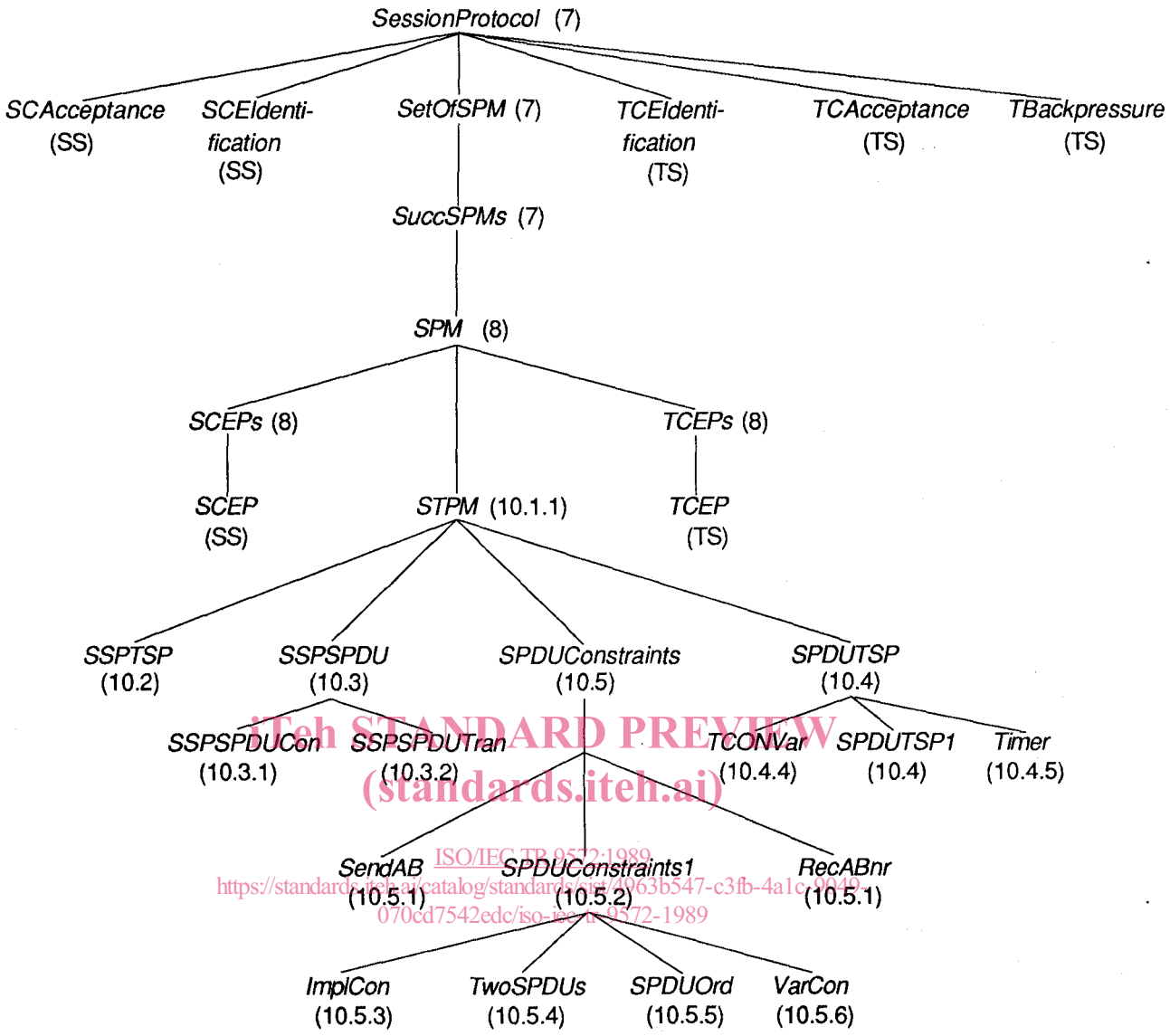

Figure 4 - Processes related by a tree structure: each "father" process contains (or is constructed from) one or more instances of its immediate "descendant" process(es)

7 Overall constraints of the session protocol

A number of standard types are imported from the LOTOS library of data types by means of the *library* construct. The specification is parameterized with a protocol implementation parameter, defined in type *SProtocolImplementationPar*. Its value indicates: 1) whether or not use of the transport expedited service is implemented, 2) what is the maximum TSDU size for both directions of data flow, 3) which functional units have been implemented and 4) whether or not extended concatenation is implemented.

The top level structure shows a full synchronization of process *SetOfSPM* with two behaviour expressions which mutually interleave, viz. *SCEIdentification* synchronized with *SCAcceptance* on the one hand, and *TCEIdentification* synchronized with *TCAcceptance* and *TBackpressure* on the other. *SetOfSPM* defines the dynamic behaviour of a potentially infinite number of independent SPMs, however disregarding any concerns related to the limited capacity of the local system and the TS-provider and the need for unique connection identification. The latter concerns are addressed by the two behaviour expressions mentioned. They formulate the overall constraints that relate to a single service

boundary; the processes representing these constraints are defined in the Session Service formal description, IS/IEC/TR 9571, and the Transport Service formal description, ISO/IEC/TR 10023, respectively.

SetOfSPM represents concurrency by multiple instances of *SuccSPMs*, where each instance describes a succession of SPM behaviours. A SPM is represented by a distinct instance of *SPM*. The behaviour described by *SPM* is that of using at most one TC for support of a single SC or multiple successive SCs (based on the same TC). Termination of a TC enforces successful termination of the associated instance of *SPM*.

```
----------------------------------------------------------------------------------------------------------*)
specification SessionProtocol [s,t] (spei:SPEImplementation): noexit

library Boolean,Element, Set, String, NatRepresentations, OctetString, NaturalNumber, Bit, FBoolean,
      Octet, DecNatRepr, BitString, BitNatRepr, DecString , DecDigit
endlib

type SProtocolImplementationPar is SRequirements, TSDUSize
sorts SPEImplementation
opns
SPEImpl: Bool, SRqms, TSDUSize, Bool -> SPEImplementation
MaxTSize: SPEImplementation -> TSDUSize          ExtConc: SPEImplementation -> Bool
eqns forall tex,ExtConc:Bool, sfus:SRqms, tsduSize:TSDUSize
ofsort TSDUSize          MaxTSize(SPEImpl(tex,sfus,tsduSize,ExtConc)) = tsduSize;
ofsort Bool              ExtConc(SPEImpl(tex,sfus,tsduSize,ExtConc)) = ExtConc;
endtype

behaviour
(     ( SCEIdentification[s] || SCAcceptance[s] ) |||
      ( TCEIdentification[t] || TCAcceptance[t] || TBackpressure[t] )
)     || SetOfSPM[s,t](spei)
where

process SetOfSPM[s,t](spei:SPEImplementation): noexit := SuccSPMs[s,t](spei) |||  SetOfSPM[s,t](spei)
where
process SuccSPMs[s,t](spei:SPEImplemtation): noexit := SPM[s,t](spei) >> SuccSPMs[s,t](spei) endproc
endproc (* SetOfSPM *)
(*----------------------------------------------------------------------------------------------------------
```


8 Constraints for a single session protocol machine

Three separate constraints are distinguished with respect to the behaviour of a single SPM, namely constraints that are local to the SS boundary, constraints that are local to the TS boundary, and constraints that relate the behaviour at one boundary to that at the other. These constraints are represented by *SCEPs*, *TCEPs* and *STPM*, respectively.

SCEPs is described as the choice of two instances of *SCEP* that apply to the interaction with the Calling SS-user and with the Called SS-user, respectively, followed by either successful termination or another instance of *SCEPs*. The recursion applies only when the SPM re-uses the TC. Similarly, *TCEPs* is described as the choice of two instances of *TCEP* that apply to the interaction with the Calling TS-user and with the Called TS-user, respectively. The definitions of *SCEP* and *TCEP* are imported from ISO/IEC/TR 9571 and ISO/IEC/TR 10023 respectively.

STPM defines the mapping of SSPs to TSPs during the provision of one SC, or multiple successive SCs, supported by a single TC.

```
----------------------------------------------------------------------------------------------------------------------*)
process SPM[s,t](spei:SPEImplementation): noexit :=
SCEPs[s] |[s]| STPM[s,t](spei) |[t]| TCEPs[t]
where
process SCEPs[s]: exit := ( SCEP[s](calling) [] SCEP[s](called) ) >> ( exit [] SCEPs[s] ) endproc
process TCEPs[t]: exit := TCEP[t](CallingRole) [] TCEP[t](CalledRole) endproc
endproc (* SPM *)
(*----------------------------------------------------------------------------------------------------------------------
```

9 Session protocol data types

The Session Protocol data types consist of definitions for the construction of a SPDU (see 9.1), some session variables (see 9.2) for the introduction of some additional functions related to SSPs and TSPs (see 9.3), and for the construction of components of the internal event structure (see 9.5). A number of auxiliary definitions of general use are presented in 9.4.

9.1 Session protocol data unit

Two sets of definitions are distinguished for the construction of a SPDU: one for an "abstract" SPDU and the other for an encoded SPDU. This division is consistent with the observation that the elements of procedure related to SPDUs can be described independently of the encoding of SPDUs (i.e., the mapping into TS data units).

9.1.1 Abstract SPDU

The specification of the abstract SPDU type is accomplished by way of a number hierarchical type definitions. First the basic construction of SPDUs is presented (see 9.1.1.1), then a classification of SPDUs (see 9.1.1.2), and subsequently a number of additional functions on SPDUs (see 9.1.1.3 through 9.1.1.5) Concatenation of SPDUs is defined in 9.1.2; (groups of) SPDU parameters are defined in 9.1.3 (in as far as not defined in ISO/IEC/TR 9571).

9.1.1.1 SPDU basic construction

Type *BasicASPDU* defines functions, referred to as "constructor" functions, that yield (abstract) SPDU values. For each class, or "type", of SPDU a corresponding constructor function is defined with values of the parameters of that SPDU as arguments. Additionally, a function *DUM* is introduced for conveniently specifying error reporting. Definitions that relate to SPDU parameters are imported by *BasicASPDU* (most of them indirectly, by importing type *SessionServicePrimitive*).

```
----------------------------------------------------------------------------------------------------------------------*)
type BasicASPDU is SessionServicePrimitive, SPReference, CAItem, TDISPar, Prepare,EncItem
sorts ASPDU
opns
CN: SPReference,CAItem,SFUs,SAddress,SAddress,SData -> ASPDU
AC: SPReference,CAItem,STokens,SFUs,SAddress,SAddress,SData -> ASPDU
RF: SPReference,TDISPar,SFUs,Nat (*Version*) ,DatOctStr (*Reason Code*) -> ASPDU
FN: TDISPar,SData -> ASPDU                          DN,NF: SData -> ASPDU
AB: TDISPar,Nat (*Reason Code*),DatOctStr (*Reflect Parameter Values*), SData -> ASPDU
GTC,GTA,AA,AIA,ADA: -> ASPDU                        DT,TD: EncIt,SData -> ASPDU
CD,EX,CDA: SData -> ASPDU                           GT: STokens (*Token Item parameter*) ->  ASPDU
PT: STokens,SData -> ASPDU                          MIP: SSyncType,SSPSN,SData -> ASPDU
MIA: SSPSN,SData -> ASPDU                           MAP,AE,MAA,AEA: SSPSN,SData -> ASPDU
RA: STsAss,SSPSN,SData -> ASPDU                     RS : STsAss,SResynType,SSPSN,SData -> ASPDU
PR: PrepType -> ASPDU                               AS: SActId,SData -> ASPDU
```

```
AR: SCRef,SActId,SSPSN,SActId,SData -> ASPDU
AI,AD: Nat (*Reason Code*) -> ASPDU          DUM: DatOctStr -> ASPDU
ER : DatOctStr -> ASPDU                      ED : Nat (* Reason Code *), SData -> ASPDU
endtype
(*-----------------------------------------------------------------------------------------------
```

9.1.1.2 SPDU classification

Type *ASPDUConstant* defines a classification of SPDUs. Each class of SPDU is represented by a constant with a name similar to the abbreviated name to denote the SPDU in table 39 of ISO 8327. The auxiliary function *h* that maps these constants to natural numbers is defined in order to simplify the definition of equality on SPDU classes.

Type *ASPDUClassifiers* is a functional enrichment of *BasicASPDU*. It defines functions, referred to a "recognizer" functions, that determine whether a given SPDU is of a certain class.

```
-----------------------------------------------------------------------------------------------*)
type ASPDUConstant is NaturalNumber
sorts ASPDUConstant
opns
cn,ac,rf,fn,dn,nf,ab,aa,dt,ex,td,cd,gt,pt,gtc,gta,mip,mia,map,maa,rs,pr,ra,er,ed,as,ar,ad,ai,ada,aia,ae,aea,
      dum: -> ASPDUConstant
h: ASPDUConstant -> Nat
_eq_ , _ne_: ASPDUConstant,ASPDUConstant -> Bool
eqns forall x,y:ASPDUConstant
ofsort Nat
h(cn) = 0;                       h(ac) = Succ(0);                 h(rf) = Succ(h(ac));
h(fn) = Succ(h(rf));             h(dn) = Succ(h(fn));             h(nf) = Succ(h(dn));
h(ab) = Succ(h(nf));             h(aa) = Succ(h(ab));             h(dt) = Succ(h(aa));
h(ex) = Succ(h(dt));             h(td) = Succ(h(ex));             h(cd) = Succ(h(td));
h(cda) = Succ(h(cd));            h(gt) = Succ(h(cda));            h(pt) = Succ(h(gt));
h(gtc) = Succ(h(pt));            h(gta) = Succ(h(gtc));           h(mip) = Succ(h(gta));
h(mia) = Succ(h(mip));           h(map) = Succ(h(mia));           h(maa) = Succ(h(map));
h(rs) = Succ(h(maa));            h(pr) = Succ(h(rs));             h(ra) = Succ(h(pr));
h(er) = Succ(h(ra));             h(ed) = Succ(h(er));             h(as) = Succ(h(ed));
h(ar) = Succ(h(as));             h(ad) = Succ(h(ar));             h(ai) = Succ(h(ad));
h(ada) = Succ(h(ai));            h(aia) = Succ(h(ada));           h(ae) = Succ(h(aia));
h(aea) = Succ(h(ae));            h(dum) = Succ(h(aea));
ofsort Bool
x eq y = h(x) eq h(y);           x ne y = h(x) ne h(y);
endtype

type ASPDUClassifiers is BasicASPDU, ASPDUConstant
opns
IsCN,IsAC,IsRF,IsFN,IsDN,IsNF,IsAB,IsGTC,IsGTA,IsAA,IsDT,IsTD,IsCD,IsEX,IsCDA,IsGT,IsPT,IsMIP,
IsMIA,IsMAP,IsMAA,IsAEA,IsRS,IsRA,IsPR,IsAS,IsAR,IsAI,IsAD,IsAIA,IsADA,IsAE,IsCAT21,IsACK,IsPAB,
IsER,IsED,IsDUM,IsPR_RS,IsPR_RA,IsPR_MAA,IsACT: ASPDU -> Bool
k: ASPDU -> ASPDUConstant
eqns forall spr:SPReference, sreq:SFUs, cait:CAItem, cg,cd:SAddress, d:SData, tk:STokens,
      tdpar:TDISPar, vs,reas:Nat, rcode,rpv,err:DatOctStr, sn:SSPSN, a:STsAss, et:EncIt, sntp:SSyncType,
      rt:SResynType, pt:PrepType, aid1,aid2:SActId, scr:SCRef, pd:ASPDU
ofsort ASPDUConstant
k(CN(spr,cait,sreq,cg,cd,d)) = cn;                 k(AC(spr,cait,tk,sreq,cg,cd,d)) = ac;
k(RF(spr,tdpar,sreq,vs,rcode)) = rf;
k(FN(tdpar,d)) = fn;             k(DN(d)) = dn;                   k(NF(d)) = nf;
k(AB(tdpar,reas,rpv,d)) = ab;    k(GTC) = gtc;                    k(GTA) = gta;
k(AA) = aa;                      k(AIA) = aia;                    k(ADA) = ada;
k(DT(et,d)) = dt;                k(TD(et,d)) = td;                k(CD(d)) = cd;
k(EX(d)) = ex;                   k(CDA(d)) = cda;                 k(GT(tk)) = gt;
```

```
k(PT(tk,d)) = pt;          k(MIP(sntp,sn,d)) = mip;   k(MIA(sn,d)) = mia;
k(MAP(sn,d)) = map;        k(AE(sn,d)) = ae;          k(MAA(sn,d)) = maa;
k(AEA(sn,d)) = aea;        k(RA(a,sn,d)) = ra;        k(RS(a,rt,sn,d)) = rs;
k(PR(pt)) = pr;            k(AS(aid1,d)) = as;        k(AR(scr,aid1,sn,aid2,d)) = ar;
k(AI(reas)) = ai;          k(AD(reas)) = ad;          k(ER(err)) = er;
k(ED(reas,d)) = ed;        k(DUM(err)) = dum
ofsort Bool
IsCN(pd) = k(pd) eq cn;    IsAC(pd) = k(pd) eq ac;    IsRF(pd) = k(pd) eq rf;
IsFN(pd) = k(pd) eq fn;    IsNF(pd) = k(pd) eq nf;    IsDN(pd) = k(pd) eq dn;
IsAB(pd) = k(pd) eq ab;    IsGTC(pd) = k(pd) eq gtc;  IsGTA(pd) = k(pd) eq gta;
IsAA(pd) = k(pd) eq aa;    IsDT(pd) = k(pd) eq dt;    IsCDA(pd) = k(pd) eq cda;
IsTD(pd) = k(pd) eq td;    IsCD(pd) = k(pd) eq cd;    IsEX(pd) = k(pd) eq ex;
IsMIP(pd) = k(pd) eq mip;  IsGT(pd) = k(pd) eq gt;    IsPT(pd) = k(pd) eq pt;
IsMIA(pd) = k(pd) eq mia;  IsMAP(pd) = k(pd) eq map;  IsMAA(pd) = k(pd) eq maa;
IsAEA(pd) = k(pd) eq aea;  IsRS(pd) = k(pd) eq rs;    IsRA(pd) = k(pd) eq ra;
IsPR(pd) = k(pd) eq pr;    IsER(pd) = k(pd) eq er;    IsED(pd) = k(pd) eq ed;
IsAS(pd) = k(pd) eq as;    IsAR(pd) = k(pd) eq ar;    IsAI(pd) = k(pd) eq ai;
IsAD(pd) = k(pd) eq ad;    IsAIA(pd) = k(pd) eq aia;  IsADA(pd) = k(pd) eq ada;
IsAE(pd) = k(pd) eq ae;    IsDUM(pd) = k(pd) eq dum;
IsACK(pd) = (IsMIA(pd) or IsMAA(pd) or IsAEA(pd) or IsRA(pd) or IsAIA(pd) or IsADA(pd) or IsCDA(pd));
IsCAT21(pd) = (IsRS(pd) or IsRA(pd) or IsAD(pd) or  IsADA(pd) or IsAI(pd) or IsAIA(pd) or IsER(pd) or
      IsED(pd) or IsCD(pd) or IsCDA(pd));
not(IsAB(pd)) => IsPAB(pd) = false;
IsPAB(AB(tdpar,reas,rpv,d)) =
      ((reas eq Succ(Succ(0))) and (Length(rpv) le NatNum(Dec(9)))) or
      ((reas eq NatNum(Dec(3))) and Empty(rpv)) and Empty(d);
IsPR_RS(PR(pt)) = pt eq res;
IsPR_RA(PR(pt)) = pt eq rack;
IsPR_MAA(PR(pt)) = pt eq mack;
IsACT(pd) = IsAR(pd) or IsAS(pd) or IsAI(pd) or IsAD(pd) or IsAE(pd) or IsAIA(pd) or IsADA(pd) or
      sAEA(pd);
endtype
(*----------------------------------------------------------------------------------------------------
```


9.1.1.3 SPDU parameter selectors

Type *ASPDUParameterSelectors* defines functions, referred to "extractor" functions, that allow to determine the value of individual SPDU parameters. It imports *SLokalTokensState* from ISO/IEC/TR 9571 and *Compare*, an auxiliary type (see 9.4.2).

```
----------------------------------------------------------------------------------------------------*)
type ASPDUParameterSelectors is ASPDUClassifiers,SLocalTokensState,Compare
opns
ErrStr,RPV: ASPDU-> DatOctStr          SPSN: ASPDU -> SSPSN
SPRef: ASPDU -> SPReference            CgAddr,CdAddr: ASPDU -> SAddress
Version: ASPDU -> Nat                  ResynType: ASPDU -> SResynType
TDISPar: ASPDU -> TDISPar              Reason: ASPDU -> Nat
Preptype: ASPDU -> PrepType            MtsRecFlow,MtsSendFlow: ASPDU -> Nat
MTSDUPar: ASPDU -> TSDUSize            UserInf: ASPDU -> SData
FUs: ASPDU -> SFUs                     EncItem: ASPDU -> EncIt
TokenSet,InvTokenSet: ASPDU -> STsAss  InitialAss: ASPDU -> SLTsState
Tokens: ASPDU -> STokens               ProtOptions: ASPDU -> Bool
```

```
eqns forall sn:SSPSN, cait:CAItem, spr:SPReference, vs:Nat, sdat:SData,prt:PrepType, rpv:DatOctStr,
      srqms:SFUs, sa1,sa2:SAddress, srf:SCRef, mt,mtr,mts:Nat, tkass:STsAss, rtyp:SResynType,
      tsize:TSDUSize, actid1,actid2:SActId, reas:Nat,str:DatOctStr, tdpar:TDISPar, encit:EncIt,
      stok,CgTokens,CdTokens:STokens, tk:SToken, styp:SSyncType
ofsort DatOctStr
ErrStr(DUM(str)) = str;
ofsort SSPSN
SPSN(CN(spr,cait,srqms,sa1,sa2,sdat)) = SSPSN(cait);
SPSN(AC(spr,cait,stok,fus,sa1,sa2,sdat)) = SSPSN(cait);
SPSN(MIP(styp,sn,sdat)) = sn;          SPSN(MIA(sn,sdat)) =sn;
SPSN(MAP(sn,sdat)) =sn;    SPSN(MAA(sn,sdat)) =sn;
SPSN(AEA(sn,sdat)) =sn;    SPSN(AE(sn,sdat)) = sn;
SPSN(RS(tkass,rtyp,sn,sdat)) = sn;
SPSN(RA(tkass,sn,sdat)) = sn;          SPSN(AR(srf,actid1,sn,actid2,sdat)) = sn;
IsAS(pd) => SPSN(pd) = s(Succ(0)); (* s maps a natural number on a value of sort SSPSN *)
ofsort SPReference
SPRef(CN(spr,cait,srqms,sa1,sa2,sdat)) = spr;
SPRef(AC(spr,cait,stok,srqms,sa1,sa2,sdat)) = spr;
ofsort SAddress
CgAddr(CN(spr,cait,srqms,sa1,sa2,sdat)) = sa1;
CdAddr(CN(spr,cait,srqms,sa1,sa2,sdat)) = sa2;
CgAddr(AC(spref,cait,stok,srqms,sa1,sa2,sdat)) = sa1;
CdAddr(AC(spr,cait,stok,srqms,sa1,sa2,sdat)) = sa2;
ofsort Nat
Version(CN(spr,cait,srqms,sa1,sa2,sdat)) = Version(cait);
Version(AC(spr,cait,stok,srqms,sa1,sa2,sdat)) = Version(cait);
ofsort SResynType
ResynType(RS(tkass,rtyp,sn,sdat)) = rtyp;
ofsort TDISPar
TDISPar(RF(spr,tdpar,srqms,vs,str)) = tdpar;     TDISPar(FN(tdpar,sdat)) = tdpar;
TDISPar(AB(tdpar,reas,rpv,sdat)) = tdpar;
ofsort Nat
Reason(RF(spr,tdpar,srqms,vs,str)) = ToNatNum(First(str));
Reason(AB(tdpar,reas,rpv,sdat)) = reas;          Reason(AI(reas)) = reas;
Reason(AD(reas)) = reas;    Reason(ED(reas,sdat)) = reas;
ofsort TSDUSize
MTSDUPar(CN(spr,cait,sqrms,sa1,sa2,sdat)) = TSDUSize(cait);
MTSDUPar(AC(spr,cait,stok,srqms,sa1,sa2,sdat)) = TSDUSize(cait);
ofsort Nat
MtsSendFlow(CN(spr,cait,sqrms,sa1,sa2,sdat)) = inresp(TSDUSize(cait));
MtsRecFlow(CN(spr,cait,sqrms,sa1,sa2,sdat)) = respin(TSDUSize(cait));
MtsSendFlow(AC(spr,cait,stok,sqrms,sa1,sa2,sdat)) = inresp(TSDUSize(cait));
MtsRecFlow(AC(spr,cait,stok,sqrms,sa1,sa2,sdat)) = respin(TSDUSize(cait));
ofsort SFUs
FUs(CN(spr,cait,srqms,sa1,sa2,sdat)) = srqms; FUs(AC(spr,cait,stok,srqms,sa1,sa2,sdat)) = srqms;
FUs(RF(spr,tdpar,srqms,vs,str)) = srqms;
ofsort DatOctStr
RPV(AB(tdpar,reas,str,sdat)) = str;     RPV(ER(str)) = str;
ofsort STsAss
TokenSet(CN(spr,cait,srqms,sa1,sa2,sdat)) = STsAss(cait);
TokenSet(AC(spr,cait,stok,srqms,sa1,sa2,sdat)) = STsAss(cait);
TokenSet(RS(tkass,rtyp,sn,sdat)) = tkass;
TokenSet(RA(tkass,sn,sdat)) = tkass;
InvTokenSet(CN(spr,cait,srqms,sa1,sa2,sdat)) = STsAss(Second(STsAss(cait)),First(STsAss(cait)),{});
InvTokenSet(AC(spr,cait,stok,srqms,sa1,sa2,sdat)) = STsAss(Second(STsAss(cait)),First(STsAss(cait)),{});
InvTokenSet(RS(tkass,rtyp,sn,sdat)) = STsAss(Second(tkass),First(tkass),{});
InvTokenSet(RA(tkass,sn,sdat)) = STsAss(Second(tkass),First(tkass),{});
ofsort STokens
Tokens(AC(spr,cait,stok,srqms,sa1,sa2,sdat)) = stok;      Tokens(GT(stok)) = stok;
Tokens(PT(stok,sdat)) = stok;
```