Title: Application of the proposed E-LOTOS datatype language to the description of OSI and ODP standards

Source: AFNOR

Project: WI 1.21.20.2.3

Date: November 1995

Contents

Chapter 1

Introduction

1.1 Purpose

In this document, we investigate how formal descriptions written in LOTOS could be translated to E-LOTOS. We consider the following formal descriptions:

1. The formal description of the CCR protocol, provided by NPL (National Physical Laboratory),
2. The formal description of the TP protocol, also provided by NPL (version v4),
3. The formal description of an ODP application ("metamovie") developed by Andreas Vogel,
4. The formal description of another ODP application ("video service") developed by Frank Koch,
5. A formal description of the LAPB protocol, provided by the University of Ottawa.

We mainly focus on the data parts of these descriptions, although the behaviour parts of some descriptions is also taken into account

1.2 Executable type definitions

For each of the above descriptions, we have identified the "constructor" and "non-constructor" operations. Constructors are flagged by a comment of the form "`(*! constructor *)`.

We "oriented the equations", i.e., we turned all equations into rewrite rules, switching from an equational semantics to a rewrite rule semantics.

We then removed all "equations between constructors" by introducing auxiliary operations.

The modified descriptions were automatically checked using the CÆSAR.ADT compiler developed at INRIA Rhône-Alpes. This revealed some errors, which were corrected.

1.3 The data type language chosen

For each description, we show how the type definitions could be translated into the E-LOTOS data type language, as it is defined in Annex A of ISO/IEC JTC1/SC21/WG7 N1503 (Output of the SC21 meeting in Ottawa, July 1995).

In this document, the types and modules are not fully defined; there are still several possible alternatives. We selected the so-called "first approach" defined in Section 5.3 of Annex A of ISO/IEC JTC1/SC21/WG7 N1503. We believe that this approach is the most appropriate one, because it is compatible with existing LOTOS, it is expressive enough, and allows simple, efficient implementations.

We have made an extensive use of the new constructs proposed for E-LOTOS, notably the "**case**" construct, the "**if-then-else**" construct, and the "**let**" construct.

However, by analyzing the aforementioned formal descriptions, we have been faced to expressiveness problems, which we solved by bringing two modifications to the E-LOTOS datatype language defined in Annex A of ISO/IEC JTC1/SC21/WG7 N1503, "first approach":

- We introduced a new expression "E **match** P", where E is an expression and P a pattern. This expression yields *true* iff expression E matches pattern P. This new construct generalizes and replaces the tester functions defined in Section 6.8 of Annex A of ISO/IEC JTC1/SC21/WG7 N1503.

- We introduced a new expression "**select** X **in** E", where E and X is the formal parameter ("field") of a constructor C. Expression E should have the form "$E = C(...)$". The "**select**" construct acts as projection that extracts the value of field X from the value of E. It is allowed that several constructors have a field named X, provided that all these fields have the same type. This new construct generalizes and replaces the selector functions defined in Section 6.8 of Annex A of ISO/IEC JTC1/SC21/WG7 N1503.

For all the aforementioned LOTOS examples, we never provided an E-LOTOS translation for the syntactic equality function ("`eq`") and inequality function ("`ne`"). We assume that E-LOTOS will be defined in such a way that these functions will be automatically provided. We will only define equality and inequality functions when they differ from syntactic equality and inequality.

We have also assumed that convenient notations for natural numbers (i.e., "`1`", "`2`", ... instead of "`succ (0)`", "`succ (succ (0))`", ...) were available, as well as auxiliary functions such as: the predecessor function "`pred`" (such that "`pred (0) = 0`") and the substract function.

1.4 The modules language chosen

We adopted the so-called "LOTOSPHERE proposal", which is defined in the output document of the SC21 meeting in Yokohama, July 1993. The changes only concern with concrete syntax, an lead to simple keyword substitutions, as summarized in the following table:

LOTOS keywords	E-LOTOS keywords
type	**module**
endtype	**endmod**
sorts	**type** . . . **endtype**
opns	**function** . . . **endfunc**
sortnames	**sorts**
opnnames	**opns**
renamedby	{. . .}
actualizedby	[. . .]

1.5 Organization of the document

Each of the following chapters is devoted to one formal description of the aforementioned OSI protocols or ODP applications.

For each description, we present the original Lotos source text followed by the proposed translation in E-Lotos. We use the following syntactic annotations to delimit the boundary between Lotos and E-Lotos part:

```
                 LOTOS test
(* ---------------------------------------------------------------------------
                 E-LOTOS translation
============================================================================= *)
```

Chapter 2

OSI CCR protocol

2.1 Comments

From our analysis of the formal description of OSI CCR protocol, we can make the following remarks:

- The "selector functions" are not simply used for record types, but also for union (sum of products) types. For instance, The specification of `get_ud` selector for `CSP` type is:

  ```
  ofsort  PDUqueue (* User Data *)
          get_ud(CBeginReq(aaimn,aais,bisn,bis,ud)) = ud;
          get_ud(CBeginInd(aaimn,aais,bisn,bis,ud)) = ud;
          get_ud(CBeginRsp(ud)) = ud;
          get_ud(CBeginCnf(ud)) = ud;
          get_ud(CPrepareReq(ud)) = ud;
          get_ud(CPrepareInd(ud)) = ud;
          get_ud(CReadyReq(ud)) = ud;
          get_ud(CReadyInd(ud)) = ud;
          get_ud(CCommitReq(ud)) = ud;
          get_ud(CCommitInd(ud)) = ud;
          get_ud(CCommitRsp(ud)) = ud;
          get_ud(CCommitCnf(ud)) = ud;
          get_ud(CRollbackReq(ud)) = ud;
          get_ud(CRollbackInd(ud)) = ud;
          get_ud(CRollbackRsp(ud)) = ud;
          get_ud(CRollbackCnf(ud)) = ud;
          get_ud(CRecoverReq(rs,aaimn,aais,bisn,bis,ud)) = ud;
          get_ud(CRecoverInd(rs,aaimn,aais,bisn,bis,ud)) = ud;
          get_ud(CRecoverRsp(rr,aaimn,aais,bisn,bis,ud)) = ud;
          get_ud(CRecoverCnf(rr,aaimn,aais,bisn,bis,ud)) = ud;
          get_ud(CCommitReq_CBeginReq(ud,aaimn,aais,bisn,bis,udb)) = ud;
          get_ud(CRollbackReq_CBeginReq(ud,aaimn,aais,bisn,bis,udb)) = ud;
  ```

  These equations could be directly translated in E-Lotos using pattern matching and "case" statements:

  ```
  function get_ud(csp:CSP) : PDUqueue is
    case csp in
  ```

```
            CBeginReq(UD:=UD:PDUqueue,...)
          | CBeginInd(UD:=UD:PDUqueue,...)
          | CBeginRsp(UD:PDUqueue)
          | CBeginCnf(UD:PDUqueue)
          | CPrepareReq(UD:PDUqueue)
          | CPrepareInd(UD:PDUqueue)
          | CReadyReq(UD:PDUqueue)
          | CReadyInd(UD:PDUqueue)
          | CCommitReq(UD:PDUqueue)
          | CCommitInd(UD:PDUqueue)
          | CCommitRsp(UD:PDUqueue)
          | CCommitCnf(UD:PDUqueue)
          | CRollbackReq(UD:PDUqueue)
          | CRollbackInd(UD:PDUqueue)
          | CRollbackRsp(UD:PDUqueue)
          | CRollbackCnf(UD:PDUqueue)
          | CRecoverReq(UD:=UD:PDUqueue,...)
          | CRecoverInd(UD:=UD:PDUqueue,...)
          | CRecoverRsp(UD:=UD:PDUqueue,...)
          | CRecoverCnf(UD:=UD:PDUqueue,...)
          | CCommitReq_CBeginReq(UD:=UD:PDUqueue,...)
          | CRollbackReq_CBeginReq(UD:=UD:PDUqueue,...) -> UD
  endcase
endfunc
```

However, the "**select**" operator that we introduced makes the translation much shorter:

```
function get_ud(csp:CSP) : PDUqueue is
  select UD in CSP
endfunc
```

- We can notice the existence of overloaded constructors in the CCR description, as for instance:

```
type    SP is
          sp (csp: CSP)
        | sp (asp: ASP)
        | sp (psp: PSP)
endtype
```

The "**select**" statement is able to deal with overloaded constructors:

```
function get_csp (sp:SP) : CSP is select csp in sp endfunc
function get_asp (sp:SP) : ASP is select asp in sp endfunc
function get_psp (sp:SP) : PSP is select psp in sp endfunc
```

- The CCR description contains many auxiliary functions, for instance constant functions and "h" functions, which are used to define equality, inequality, and tester functions. For instance, in type `Responder_Recovery_State`, we have:

```
        h : responder_recovery_state -> key
        _eq_,_ne_ : responder_recovery_state,responder_recovery_state -> Bool
ofsort  key
        h(done) = 0;
        h(unknown) = succ(h(done));
        h(retry) = succ(h(unknown));
ofsort  Bool
        r eq r1 = h(r) eq h(r1);
        r ne r1 = not(r eq r1);
```

There are similar "h" functions defined in types `Responder_Recovery_State`, `Result`, `Result_Source`, `Sync_Type`, `ReSync_Type`, `CCR_Service_Primitive`, `ACSE_Service_Primitive`, `Presentation_Service_Primitive`, `CCR_PDU`, `Sub_A_PDU`, `Protocol_Data_Unit`, `PDU_Queue`, `CCR_Version_Number`.

The functions are used to assign a unique number to each constructor these types. In E-LOTOS, the "**h**" functions and the constants could certainly be avoided because:

– syntactical equality is automatically generated for (equality) types,

– tester functions are obtained using the "**match**" construct

In some cases, some "h" functions are used in the behaviour part of the CCR protocol. However, this could be avoided by writing expressions in a different way. A very common example is the the guard:

```
[h(ccr_pdu(pdu_q)) eq c_begin_ri ]
```

which could be replaced with:

```
[is_c_begin_ri (ccr_pdu(pdu_q))]
```

There are more complex examples, for instance the guards:

```
[may_conc(h(pdu_q),h(pdu))]
```

and

```
[not(may_conc(h(pdu_q),h(pdu)))]
```

For this example, we have provided a translation of function "`may_conc`" that avoids to use function "`h`" in the behaviour part.

Moreover, in type `CCR_Service_Primitive`, functions constants (like `CBegin_req`) could be eliminated because they are only used to define tester functions.

- Also in behaviour part, we could use "`case`" constructs and pattern-matching to avoid the use of tester functions, as in the following example:

```
case ccr_pdu(pdu_q) in
   c_begin_ri(...) -> ( [ver eq ver1]->
            ( p!PSyncMinReq(opt,s_null,pdu_q,off);
              LowerMapper[l_u,l_l,p,u_env,ver](ver,false)
            )
            ...)
| c_begin_rc(...) -> ( [c_r_b eq false]->
          ( u_env?s_p:sync_point;
            p!PSyncMinRsp(s_p,pdu_q);
            LowerMapper[l_u,l_l,p,u_env,ver](ver,c_r_b)
          )
          ...)
| c_prepare_ri(...) -> ( p!PTypedDataReq(pdu_q);
          LowerMapper[l_u,l_l,p,u_env,ver](ver,c_r_b)
```

```
                )
         | c_ready_ri(...) ->
         ...
         endcase
```

Using this approach, we no longer need the "`key`" types.

2.2 Formal description in both LOTOS and E-LOTOS

```
(*--------------------------------------------------------------*)
(*                                                              *)
(*      LOTOS Description of CCR Protocol Specification         *)
(*                                                              *)
(* Modified by Radu Mateescu: May 1995                          *)
(* - equations between constructors removed                     *)
(* Modified by Mihaela Sighireanu: November 1995                *)
(* - translation in E-LOTOS provided                            *)
(*--------------------------------------------------------------*)

(*--------------------------------------------------------------*)
(*      The Structure of LOTOS Description of CCR               *)
(*--------------------------------------------------------------*)
(* Diagrams or something to be provided *)

(*--------------------------------------------------------------*)
(*      CCR Protocol Specification                              *)
(*--------------------------------------------------------------*)
specification ISO_9805_CCR_Spec[u,p,acse_u,acse_l,env,u_env]
                                              (ver:ccr_ver) : exit

(*--------------------------------------------------------------*)
(*      Library                                                 *)
(*--------------------------------------------------------------*)
library
      Boolean (*,Element *)
endlib

(*--------------------------------------------------------------*)
(*      Type Specification                                      *)
(*--------------------------------------------------------------*)
(*--------------------------------------------------------------*)
(*      General Type :                                          *)
(*      This type is a general type. There are some types       *)
(*      which have a structure, but whose structure is not      *)
(*      important in this LOTOS description. In such a case,    *)
(*      this type is used as a place holder.                    *)
(*--------------------------------------------------------------*)
type    General is Boolean
sorts   general
opns
        null (*! constructor *) : -> general
        succ (*! constructor *) : general -> general
        _eq_,_ne_ : general,general -> Bool
```

```
eqns
forall  x,y:general
ofsort  Bool
        null eq null = true;
        null eq succ(x) = false;
        succ(x) eq null = false;
        succ(x) eq succ(y) = x eq y;
        x ne y = not(x eq y);
endtype (* General *)
(* -----------------------------------------------------------------------
module  General is  Boolean
type    general is
          null
        | succ (general)
endtype
endmod
======================================================================== *)
(*--------------------------------------------------------------*)
(*      Key Type :                                              *)
(*      This type is used for identification which is necessary *)
(*      in other types.                                         *)
(*--------------------------------------------------------------*)
type    Key is Boolean
sorts   key
opns
        0 (*! constructor *) : -> key
        succ (*! constructor *) : key -> key
        _eq_,_ne_,_le_ : key,key -> Bool
eqns
forall  x,x1,y,y1:key
ofsort  Bool
        0 eq 0 = true;
        0 eq succ(x) = false;
        succ(x) eq 0 = false;
        succ(x) eq succ(y) = x eq y;
        x ne y = not(x eq y);
        0 le 0 = true;
        0 le succ(x) = true;
        succ(x) le succ(y) = x le y;
        succ(x) le 0 = false;
endtype (* Key *)
(* -----------------------------------------------------------------------
module  Key is Boolean
type    key is
          0
        | succ(key)
endtype

function _le_ (K1:key,K2:key) : Bool
  case K1 in
          0      -> true
        | succ(X:general)
                 -> case K2 in
                         0            -> false
```

```
                    | succ(Y:general) -> X le Y
                endcase
  endcase
endfunc
endmod
========================================================================== *)
(*-----------------------------------------------------------*)
(*      Formal Identifier Type :                             *)
(*      This type is a formal type for the definition of     *)
(*      "Atomic Action Identifier" and "Branch Identifier".  *)
(*      These two types have the same structure. Therefore,  *)
(*      this formal type is useful for the definitions as    *)
(*      a template.                                          *)
(*-----------------------------------------------------------*)
(* type F_Name is Element renamedby
sortnames
        f_name for Element
endtype *)      (* F_Name *)
(* -----------------------------------------------------------------------
module  F_Name is Element {sorts f_name for Element}
endmod
========================================================================== *)

(* type F_Suffix is Element renamedby
sortnames
        f_suffix for Element
endtype *) (* F_Suffix *)
(* -----------------------------------------------------------------------
module  F_Suffix is Element {sorts f_suffix for Element}
endmod
========================================================================== *)

(* type F_Identifier is F_Name,F_Suffix
sorts
        f_identifier
opns
        f_identifier : f_name,f_suffix -> f_identifier
        get_fn : f_identifier -> f_name
        get_fs : f_identifier -> f_suffix
        _eq_,_ne_ : f_identifier,f_identifier -> FBool
eqns
forall  fn,fn1:f_name,fs,fs1:f_suffix,f,f1:f_identifier
ofsort  f_name
        get_fn(f_identifier(fn,fs)) = fn;
ofsort  f_suffix
        get_fs(f_identifier(fn,fs)) = fs;
ofsort  FBool
        f eq f1 = (get_fn(f) eq get_fn(f1)) and (get_fs(f) eq get_fs(f1));
        f ne f1 = not(f eq f1);
endtype *)      (* F_Identifier *)
(* -----------------------------------------------------------------------
module  F_Identifier is F_Name, F_Suffix
type f_identifier is
        f_identifier (FN:f_name, FS:f_suffix)
```

```
endtype

function get_fn(FI:f_identifier) : f_name is select FN in FI endfunc

function get_fs(FI:f_identifier) : f_suffix is select FS in FI endfunc
endmod
========================================================================== *)
(*-------------------------------------------------------------*)
(*      Atomic Action Identifier :                             *)
(*      This type consists of the pair of "Masters Name" and    *)
(*      "Atomic Action Suffix". Formal Identifier Type is used. *)
(*-------------------------------------------------------------*)
(* type F_Atomic_Action_Identifier is F_Identifier renamedby
sortnames
        atomic_action_identifier for f_identifier
opnnames
        atomic_action_identifier for f_identifier
        get_mn for get_fn
        get_aas for get_fs
endtype *) (* F_Atomic_Action_Identifier *)
(* ------------------------------------------------------------------------
module  F_Atomic_Action_Identifier is
        F_Identifier {sorts atomic_action_identifier for f_identifier
                  opns  atomic_action_identifier for f_identifier
                        get_mn                   for get_fn
                        get_aas                  for get_fs }
endmod
========================================================================== *)
(*
type    Atomic_Action_Identifier is F_Atomic_Action_Identifier actualizedby
                        Atomic_Action_Suffix,Masters_Name,Boolean using
sortnames
        masters_name for f_name
        atomic_action_suffix for f_suffix
        Bool for FBool
endtype *) (* Atomic_Action_Identifier *)
(* ------------------------------------------------------------------------
module  Atomic_Action_Identifier is
        F_Atomic_Action_Identifier
        [Atomic_Action_Suffix + Masters_Name + Boolean ::
        sorts   masters_name for f_name
                atomic_action_suffix for f_suffix
                Bool for FBool ]
endmod
========================================================================== *)
(*-------------------------------------------------------------*)
(*      This type is an instanciation "by hand" of the          *)
(*      F_Atomic_Action_Identifier type.                        *)
(*-------------------------------------------------------------*)
type    Atomic_Action_Identifier is Atomic_Action_Suffix, Masters_Name, Boolean
sorts
        atomic_action_identifier
opns
        atomic_action_identifier (*! constructor *)
```

```
                : masters_name, atomic_action_suffix -> atomic_action_identifier
        get_mn : atomic_action_identifier -> masters_name
        get_aas : atomic_action_identifier -> atomic_action_suffix
        _eq_, _ne_ : atomic_action_identifier, atomic_action_identifier -> Bool
eqns
forall
        mn : masters_name, aas : atomic_action_suffix,
        aai, aai1 : atomic_action_identifier
ofsort  masters_name
        get_mn (atomic_action_identifier (mn, aas)) = mn;
ofsort  atomic_action_suffix
        get_aas (atomic_action_identifier (mn, aas)) = aas;
ofsort Bool
        aai eq aai1 = (get_mn (aai) eq get_mn (aai1)) and
                      (get_aas (aai) eq get_aas (aai1));
        aai ne aai1 = not (aai eq aai1);
endtype (* Atomic_Action_Identifier *)
(* -----------------------------------------------------------------------
module  Atomic_Action_Identifier is Atomic_Action_Suffix,
                                    Masters_Name, Boolean
type    atomic_action_identifier is
        atomic_action_identifier(MN:masters_name, AAS:atomic_action_suffix)
endtype

function get_mn(AAI:atomic_action_identifier) : masters_name is
        select MN in AAI
endfunc

function get_fs(FI:atomic_action_identifier) : atomic_action_suffix is
        select AAS in FI
endfunc
endmod
======================================================================== *)
(*-------------------------------------------------------------*)
(*      Branch Identifier :                                    *)
(*      This type consists of the pair of "Superiors Name" and  *)
(*      "Branch Suffix". Formal Identifier Type is used.        *)
(*-------------------------------------------------------------*)
(* type F_Branch_Identifier is F_Identifier renamedby
sortnames
        branch_identifier for f_identifier
opnnames
        branch_identifier for f_identifier
        get_sn for get_fn
        get_bs for get_fs
endtype *) (* F_Branch_Identifier *)
(* -----------------------------------------------------------------------
module  F_Branch_Identifier is
        F_Identifier {sorts branch_identifier for f_identifier
                     opns  branch_identifier for f_identifier
                           get_sn for get_fn
                           get_bs for get_fs }
endmod
======================================================================== *)
```

```
(* type Branch_Identifier is F_Branch_Identifier actualizedby
                                Branch_Suffix,Superiors_Name,Boolean using
sortnames
        superiors_name for f_name
        branch_suffix for f_suffix
        Bool for FBool
endtype *)(* Branch_Identifier *)
(* -----------------------------------------------------------------------
module  Branch_Identifier is
        F_Branch_Identifier [Branch_Suffix + Superiors_Name + Boolean ::
                                sorts superiors_name for f_name
                                      branch_suffix for f_suffix
                                      Bool for FBool]
endmod
======================================================================= *)
(*-------------------------------------------------------------*)
(*      This type is an instanciation "by hand" of the          *)
(*      F_Branch_Identifier type.                               *)
(*-------------------------------------------------------------*)
type    Branch_Identifier is Branch_Suffix, Superiors_Name, Boolean
sorts
        branch_identifier
opns
        branch_identifier (*! constructor *)
                : superiors_name, branch_suffix -> branch_identifier
        get_sn : branch_identifier -> superiors_name
        get_bs : branch_identifier -> branch_suffix
        _eq_, _ne_ : branch_identifier, branch_identifier -> Bool
eqns
forall
        sn : superiors_name, bs : branch_suffix,
        bi, bi1 : branch_identifier
ofsort  superiors_name
        get_sn (branch_identifier (sn, bs)) = sn;
ofsort  branch_suffix
        get_bs (branch_identifier (sn, bs)) = bs;
ofsort Bool
        bi eq bi1 = (get_sn (bi) eq get_sn (bi1)) and (get_bs (bi) eq get_bs (bi1));
        bi ne bi1 = not (bi eq bi1);
endtype (* Branch_Identifier *)
(* -----------------------------------------------------------------------
module  Branch_Identifier is Branch_Suffix, Superiors_Name, Boolean
type    branch_identifier is
        branch_identifier (SN:superiors_name, BS:branch_suffix)
endtype

function get_sn(BI:branch_identifier) : superiors_name is select SN in BI endfunc
function get_bs(BI:branch_identifier) : branch_suffix is select BS in BI endfunc
endmod
======================================================================= *)
(*-------------------------------------------------------------*)
(*      Masters Name :                                          *)
(*      This type is constructed by the renaming of Name Type.  *)
(*      This type and Superiors Name Type have the same         *)
(*      structure.                                              *)
(*--------------------------------------------------------------*)
type    Masters_Name is Name renamedby
sortnames
        masters_name for name
opnnames
        masters_name for name
endtype (* Masters_Name *)
(* ---------------------------------------------------------------------------
module  Masters_Name is
        Name{sorts masters_name for name
             opns  masters_name for name}
endmod
=========================================================================== *)
(*--------------------------------------------------------------*)
(*      Superiors Name                                          *)
(*      This type is constructed from Name Type. This type and  *)
(*      Masters Name Type have the same structures.             *)
(*--------------------------------------------------------------*)
type    Superiors_Name is Name renamedby
sortnames
        superiors_name for name
opnnames
        superiors_name for name
endtype (* Superiors_Name *)
(* ---------------------------------------------------------------------------
module  Superiors_Name is
        Name{sorts superiors_name for name
             opns  superiors_name for name}
endmod
=========================================================================== *)
(*--------------------------------------------------------------*)
(*      Name Type :                                             *)
(*      This is a general Name Type, and is used for            *)
(*      the definition of Masters Name and Superiors Name.      *)
(*      This type is composed of AE Title and Side.             *)
(*--------------------------------------------------------------*)
type    Name is AE_title,Side,Boolean,CCR_Version_Number
sorts   name
opns
        name (*! constructor *) : ae_title -> name
        name (*! constructor *) : side -> name
        get_aet : name -> ae_title
        get_s : name -> side
        is_aet : name -> Bool
        is_s : name -> Bool
        _eq_,_ne_: name,name -> Bool
        get_aet: name, ccr_ver, ae_title, ae_title -> ae_title
eqns
forall  aet,sender_aet,receiver_aet:    ae_title,
        s,s1:   side,
        n,n1:   name,
        ver:    ccr_ver
ofsort  Bool
```

```
        is_aet(name(aet)) = true;
        is_aet(name(s)) = false;
        is_s(name(aet)) = false;
        is_s(name(s)) = true;
ofsort  AE_title
        get_aet(name(aet)) = aet;
        get_aet(name(aet),ver,receiver_aet,sender_aet) = aet;
        get_aet(name(receiver),ver2,receiver_aet,sender_aet) = receiver_aet;
        get_aet(name(sender),ver2,receiver_aet,sender_aet) = sender_aet;
ofsort  side
        get_s(name(s)) =s;
ofsort  Bool
        is_aet(n) and is_aet(n1) =>
                n eq n1 = get_aet(n) eq get_aet(n1);
        is_s(n) and is_s(n1) =>
                n eq n1 = get_s(n) eq get_s(n1);
        is_aet(n) and is_s(n1) =>
                n eq n1 = false;
        is_s(n) and is_aet(n1) =>       n eq n1 = false;
        n ne n1 = not(n eq n1);
endtype (* Name *)
(* -----------------------------------------------------------------------
module  Name is AE_title, Side, Boolean, CCR_Version_Number
type    name is
          name(AET:ae_title)
        | name(S:side)
endtype

function get_aet(N:name) : ae_title is select AET in N endfunc

function get_s(N:name) : side is select S in N endfunc

function get_aet(N:name,CV:ccr_ver,AET1:ae_title,AET2:ae_title) :  ae_title is
  let receiver_aet=AET1, sender_aet=AET2 in
        case N in
          name(AET:=AET:ae_title)) -> AET
        | name(S:=S:side) -> let ver2=CV in
                                case S in
                                  receiver -> receiver_aet
                                | sender   -> sender_aet
                                endcase
        endcase
endfunc

function is_aet(N:name) : Bool is N match name(AET: any ae_title) endfunc

function is_s(N:name) : Bool is N match name(S: any side) endfunc

endmod
=========================================================================== *)
(*-----------------------------------------------------------------*)
(*      AE Title Type :                                            *)
(*      AE Title is an important name in OSI Application Layer,   *)
(*      which is composed of AP Title and AE Qualifier.            *)
```

```
(*-------------------------------------------------------------*)
type    AE_title is AP_title,AE_qualifier,Boolean
sorts   ae_title
opns
        ae_null : -> ae_title
        aet (*! constructor *) : ap_title,ae_qualifier -> ae_title
        get_apt : ae_title -> ap_title
        get_aeq : ae_title -> ae_qualifier
        _eq_,_ne_ : ae_title,ae_title -> Bool
eqns
forall  apt,apt1:ap_title,aeq,aeq1:ae_qualifier,aet,aet1:ae_title
ofsort  ap_title
        get_apt(aet(apt,aeq)) = apt;
ofsort  ae_qualifier
        get_aeq(aet(apt,aeq)) = aeq;
ofsort  ae_title
        ae_null = aet(ap_null,aq_null);
ofsort  Bool
        aet(apt,aeq) eq aet(apt1,aeq1) = (apt eq apt1) and (aeq eq aeq1);
        aet ne aet1 = not(aet eq aet1);
endtype (* AE_title *)
(* ----------------------------------------------------------------------
module  AE_title is AP_title, AE_qualifier, Boolean
type    ae_title is
        aet(APT:ap_title,AEQ:ae_qualifier)
endtype

function ae_null is aet(ap_null,aq_null)
endfunc

function get_apt(AET:ae_title) : ap_title is select APT in AET endfunc

function get_aeq(AET:ae_title) : ae_qualifier is select AEQ in AET endfunc
endmod
======================================================================= *)
(*-------------------------------------------------------------*)
(*      AP Title Type :                                        *)
(*      AP Title is defined in the Technical Corrigendum of     *)
(*      OSI ACSE , which is Name defined by Directory Service   *)
(*      or ASN1 OCTET IDENTIFIER. But, in this description, its *)
(*      structure is not important. AP Title is only used as    *)
(*      a place holder.                                         *)
(*-------------------------------------------------------------*)
type    AP_title is General renamedby
sortnames
        ap_title for general
opnnames
        ap_null for null
endtype (* AP_title *)
(* ----------------------------------------------------------------------
module  AP_title is General{sorts ap_title for general
                        opns  ap_null for null}
endmod
======================================================================= *)
```

```
(*------------------------------------------------------------*)
(*      AE Qualifier Type :                                   *)
(*      AP Title is defined in the Technical Corrigendum of   *)
(*      OSI ACSE , which is RelativeDistingushedName defined by *)
(*      Directory Service or ASN1 INTEGER. But, in this       *)
(*      description, its structure is not important. AP Title  *)
(*      is only used as a place holder.                       *)
(*------------------------------------------------------------*)
type    AE_qualifier is General renamedby
sortnames
        ae_qualifier for general
opnnames
        aq_null for null
endtype (* AE_qualifier *)
(* -----------------------------------------------------------------------
module  AE_qualifier is General{sorts ae_qualifier for general
                        opns  aq_null for null}
endmod
======================================================================= *)
(*------------------------------------------------------------*)
(*      Side Type :                                           *)
(*      Side is used in CCR version 2. It includes "sender"    *)
(*      or "receiver". The notation of side depends on         *)
(*      AE Titles which are transferred in the establishment of *)
(*      an association. It is used in CCR PDU.                 *)
(*------------------------------------------------------------*)
type    Side is Boolean
sorts   side
opns
        sender(*! constructor *), receiver (*! constructor *) : -> side
        _eq_,_ne_ : side,side -> Bool
eqns
forall  s, s1 : side
ofsort  Bool
        sender eq sender = true;
        receiver eq sender = false;
        sender eq receiver = false;
        receiver eq receiver = true;
        s ne s1 = not (s eq s1);
endtype (* Side *)
(* -----------------------------------------------------------------------
module  Side is Boolean
type    side is
          sender
        | receiver
endtype
endmod
======================================================================= *)
(*------------------------------------------------------------*)
(*      Atomic Action Suffix Type :                           *)
(*      This type is a kind of identifier of Atomic Action,    *)
(*      which is more local than Atomic Action Identifier.     *)
(*      In this description, it is not important and is used as *)
(*      a place holder. It is used in CCR Service Primitive and *)
(*      CCR PDU.                                                *)
(*--------------------------------------------------------------*)
type    Atomic_Action_Suffix is General renamedby
sortnames
        atomic_action_suffix for general
endtype (* Atomic_Action_Suffix *)
(* ----------------------------------------------------------------------
module  Atomic_Action_Suffix is
        General{sorts atomic_action_suffix for general}
endmod
======================================================================= *)
(*--------------------------------------------------------------*)
(*      Branch Suffix Type :                                    *)
(*      This type is a kind of identifier of Branch, which is   *)
(*      more local than Branch Identifier. In this description, *)
(*      it is not important and is used as a place holder.      *)
(*      It is used in CCR Service Primitive and CCR PDU.        *)
(*--------------------------------------------------------------*)
type    Branch_Suffix is General renamedby
sortnames
        branch_suffix for general
endtype (* Branch_Suffix *)
(* ----------------------------------------------------------------------
module  Branch_Suffix is
        General{sorts branch_suffix for general}
endmod
======================================================================= *)
(*--------------------------------------------------------------*)
(*      Recovry State Type :                                    *)
(*      This type is a recovery state in CCR. In ASN.1,         *)
(*      this type is represented by two ENUMERATE types.        *)
(*      Therefore, in this description, two types, i.e.,        *)
(*      Requestor_Recovery_State and Responder_Recovery_State,  *)
(*      are also defined.                                       *)
(*      It is used in CCR Service Primitive and CCR PDU.        *)
(*--------------------------------------------------------------*)
type    Requestor_Recovery_State is Key
sorts   requestor_recovery_state
opns
        commit(*! constructor *),
        ready (*! constructor *) : -> requestor_recovery_state
        h : requestor_recovery_state -> key
        _eq_,_ne_ : requestor_recovery_state,requestor_recovery_state -> Bool
eqns
forall  r,r1:requestor_recovery_state
ofsort  key
        h(commit) = 0;
        h(ready) = succ(h(commit));
ofsort  Bool
        r eq r1 = h(r) eq h(r1);
        r ne r1 = not(r eq r1);
endtype (* Requestor_Recovery_State *)
(* ----------------------------------------------------------------------
module  Requestor_Recovery_State is Key
type    requestor_recovery_state is
          commit
        | ready
endtype

/* no translation is needed for function h */
endmod
========================================================================== *)
type    Responder_Recovery_State is Key
sorts   responder_recovery_state
opns
        done (*! constructor *),
        unknown (*! constructor *),
        retry (*! constructor *) : -> responder_recovery_state
        h : responder_recovery_state -> key
        _eq_,_ne_ : responder_recovery_state,responder_recovery_state -> Bool
eqns
forall  r,r1:responder_recovery_state
ofsort  key
        h(done) = 0;
        h(unknown) = succ(h(done));
        h(retry) = succ(h(unknown));
ofsort  Bool
        r eq r1 = h(r) eq h(r1);
        r ne r1 = not(r eq r1);
endtype (* Responder_Recovery_State *)
(* ------------------------------------------------------------------------
module  Responder_Recovery_State is Key
type    responder_recovery_state is
          done
        | unknown
        | retry
endtype

/* no translation is needed for function h */
endmod
========================================================================== *)
(*-------------------------------------------------------------*)
(*      Result :                                               *)
(*      This is the result of an association establishment.    *)
(*      It is used in ACSE Service Primitive.                  *)
(*-------------------------------------------------------------*)
type    Result is Key
sorts   result
opns
        accepted (*! constructor *),
        rejected_permanent (*! constructor *),
        rejected_transient (*! constructor *) : -> result
        _eq_,_ne_ : result,result -> Bool
        h : result -> key
eqns
forall  r,r1:result
ofsort  key
        h(accepted) = 0;
        h(rejected_permanent) = succ(h(accepted));
        h(rejected_transient) = succ(h(rejected_permanent));
ofsort  Bool
        r eq r1 = h(r) eq h(r1);
        r ne r1 = not(r eq r1);
endtype (* Result *)
(* ----------------------------------------------------------------------------
module  Result is Key
type    result is
          accepted
        | permanent
        | rejected
endtype

/* no translation is needed for function h */
endmod
=========================================================================== *)
(*-----------------------------------------------------------------*)
(*      Result Source :                                            *)
(*      This represents an entity (or a layer) which reports       *)
(*      the result of an association establishment.                *)
(*      It is used in ACSE Service Primitive.                      *)
(*-----------------------------------------------------------------*)
type    Result_Source is Key
sorts   result_source
opns
        ACSE_service_user (*! constructor *),
        ACSE_service_provider (*! constructor *),
        presentation_service_provider (*! constructor *) : -> result_source
        _eq_,_ne_ : result_source,result_source -> Bool
        h : result_source -> key
eqns
forall  r,r1:result_source
ofsort  key
        h(ACSE_service_user) = 0;
        h(ACSE_service_provider) = succ(h(ACSE_service_user));
        h(presentation_service_provider) = succ(h(ACSE_service_provider));
ofsort  Bool
        r eq r1 = h(r) eq h(r1);
        r ne r1 = not(r eq r1);
endtype (* Result_Source *)
(* ----------------------------------------------------------------------------
module  Result_Source is Key
type    result_source is
          ACSE_service_user
        | ACSE_service_provider
        | presentation_service_provider
endtype

/* no translation is needed for function h */
endmod
=========================================================================== *)
(*-----------------------------------------------------------------*)
(*      Diagnostic :                                               *)
(*      This represents a diagnostic about the result of        *)
(*      an association establishment.                           *)
(*      It is used in ACSE Service Primitive.                   *)
(*--------------------------------------------------------------*)
type    Diagnostic is
sorts   diagnostic
opns
        no_reason_given (*! constructor *) : -> diagnostic
endtype (* Diagnostic *)
(* ---------------------------------------------------------------------------
module  Diagnostic is
type    diagnostic is
        no_reason_given
endtype
endmod
========================================================================== *)
(*--------------------------------------------------------------*)
(*      Other Parameters                                        *)
(*      This is just a place holder, in which other parameters  *)
(*      except for the above in ACSE Service Primitive are set. *)
(*--------------------------------------------------------------*)
type    Other_Parameters is General renamedby
sortnames
        other_parameters for general
endtype (* Other_Parameters *)
(* ---------------------------------------------------------------------------
module  Other_Parameters is General{sorts other_parameters for general}
endmod
========================================================================== *)
(*--------------------------------------------------------------*)
(*      Synchronization Type :                                  *)
(*      Synchronization Type is a type of minor synchronization *)
(*      in Session Layer. "Explicit" and "Optional" are         *)
(*      included.                                               *)
(*--------------------------------------------------------------*)
type    Sync_Type is Key
sorts   sync_type
opns
        exp(*! constructor *) ,opt(*! constructor *)  : -> sync_type
        h : sync_type -> key
        _eq_,_ne_ : sync_type,sync_type -> Bool
eqns
forall  s,s1:sync_type
ofsort  key
        h(exp) = 0;
        h(opt) = succ(h(exp));
ofsort  Bool
        s eq s1 = h(s) eq h(s1);
        s ne s1 = not(s eq s1);
endtype (* Sync_Type *)
(* ---------------------------------------------------------------------------
module  Sync_Type is Key
type    sync_type is
          exp
        | opt
endtype

/* no translation is needed for function h */
endmod
========================================================================== *)
(*-------------------------------------------------------------*)
(*      Synchronization Point Serial Number :                  *)
(*      This type may be an integer actually, but it is not     *)
(*      important in this specification. Therefore, it is       *)
(*      represented by General Type.                            *)
(*-------------------------------------------------------------*)
type    Sync_Point_Number is General renamedby
sortnames
        sync_point for general
opnnames
        s_null for null
endtype (* Sync_Point_Number *)
(* ------------------------------------------------------------------------
module  Sync_Point_Number is General{sorts sync_point for general
                        opns  s_null for null}
endmod
========================================================================== *)
(*-------------------------------------------------------------*)
(*      Re-Synchronization Type :                              *)
(*      This type is the resynchronization type in Session     *)
(*      Layer. Three terms, abandon, restart and set, are      *)
(*      defined.                                               *)
(*-------------------------------------------------------------*)
type    ReSync_Type is Key
sorts   resync_type
opns
        abandon(*! constructor *),
        restart(*! constructor *),
        set(*! constructor *)  : -> resync_type
        h : resync_type -> key
        _eq_,_ne_ : resync_type,resync_type -> Bool
eqns
forall  r,r1:resync_type
ofsort  key
        h(abandon) = 0;
        h(restart) = succ(h(abandon));
        h(set) = succ(h(restart));
ofsort  Bool
        r eq r1 = h(r) eq h(r1);
        r ne r1 = not(r eq r1);
endtype (* ReSync_Type *)
(* ------------------------------------------------------------------------
module  ReSync_Type is Key
type    resync_type is
          abandon
        | restart
        | set
endtype
```

```
/* no translation is needed for function h */
endmod
========================================================================= *)
(*--------------------------------------------------------------*)
(*      Token Type :                                            *)
(*      Token is used in Session Layers. Token is an important  *)
(*      for CCR user, but unless the description of CCR user is *)
(*      detailed in a certain sense, it is difficult to         *)
(*      describe that CCR provider checks whether the service   *)
(*      requestor has the valid token or not. This description  *)
(*      specifies the concerns about CCR providers. Therfore,   *)
(*      this type is simply defined.                            *)
(*      When in this descrption token is necessary              *)
(*      (e.g. mapping CCR PDUs to presentation services),       *)
(*      the environment offers the specific values.             *)
(*--------------------------------------------------------------*)
type    Token is General renamedby
sortnames
        token for general
opnames
        t_null for null
endtype (* Token *)
(* ------------------------------------------------------------------------
module  Token is General{sorts token for general
                        opns  t_null for null}
endmod
========================================================================= *)
(*--------------------------------------------------------------*)
(*      Presentation Context Identifier List :                  *)
(*      This type is not important in this specification.       *)
(*--------------------------------------------------------------*)
type    P_Cont_Id_List is General renamedby
sortnames
        p_cont_id_list for general
opnames
        p_null for null
endtype (* P_Cont_Id_List *)
(* ------------------------------------------------------------------------
module  P_Cont_Id_List is General{sorts p_cont_id_list for general
                                opns  p_null for null}
endmod
========================================================================= *)
(*--------------------------------------------------------------*)
(*      Data Separation Type :                                  *)
(*      Data Separation is one of Session Functional Units.     *)
(*      It is only used in CCR version 2.                       *)
(*--------------------------------------------------------------*)
type    Data_Separation is Boolean renamedby
sortnames
        data_sep for Bool
opnames
        on for true
        off for false
```

```
endtype (* Data_Separation *)
(* ---------------------------------------------------------------------
module  Data_Separation is Boolean{sorts data_sep for Bool
                                  opns  on for true
                                        off for false}
endmod
======================================================================== *)
(*-------------------------------------------------------------*)
(*      CCR Protocol Data Unit User Data :                     *)
(*      This type is not important in this specification, and   *)
(*      a just place holder.                                   *)
(*-------------------------------------------------------------*)
(* type User_Data is General renamedby
sortnames
        user_data for general
endtype *)      (* User_Data *)
(* ---------------------------------------------------------------------
module  User_Data is General{sorts user_data for general}
endmod
======================================================================== *)
(*-------------------------------------------------------------*)
(*      This is a redefinition of the User_Data type, to avoid  *)
(*      the theoretical limitation due to the introduction of   *)
(*      constructor to_pdu0 for a renamed sort                 *)
(*-------------------------------------------------------------*)
type    User_Data is Boolean
sorts   user_data
opns
        null (*! constructor *) : -> user_data
        succ (*! constructor *) : user_data -> user_data
        _eq_,_ne_ : user_data,user_data -> Bool
eqns
forall  x,y:user_data
ofsort  Bool
        null eq null = true;
        null eq succ(x) = false;
        succ(x) eq null = false;
        succ(x) eq succ(y) = x eq y;
        x ne y = not(x eq y);
endtype (* User_Data *)
(* ---------------------------------------------------------------------
module  User_Data is Boolean, PDU_Queue
type   user_data is
          null
        | succ
        | to_pdu0 (PQ: PDUqueue)
endtype
endmod
======================================================================== *)
(*-------------------------------------------------------------*)
(*      Transformation Between User_Data and CCR_User_Data :    *)
(*      In this description, the user data of each service      *)
(*      Primitive is represented by PDU_Queue. It is necessary  *)
(*      for the concatenation/deconcatenation of PDUs for       *)
(*      Application Layers in Presentation Layer Service        *)
(*      Primitives. The type of user data of service primitives *)
(*      are different from that of CCR PDUs. Therefore,         *)
(*      the transformation functions between two different      *)
(*      types are necessary.                                    *)
(*--------------------------------------------------------------*)
type    Transformation_User_Data is PDU_Queue,User_Data
opns
        to_pdu0(*! constructor *): PDUqueue -> user_data
        to_sp0 (*! constructor *): user_data -> PDUqueue
        to_pdu: PDUqueue -> user_data
        to_sp : user_data -> PDUqueue
eqns
forall  q:PDUqueue,ud:user_data
ofsort  PDUqueue
        to_sp(to_pdu0(q)) = q;
        to_sp(ud) = to_sp0(ud);
ofsort  user_data
        to_pdu(to_sp0(ud)) = ud;
        to_pdu(q) = to_pdu0(q);
endtype (* Transformation_User_Data *)
(* -----------------------------------------------------------------------
module  Transformation_User_Data is PDU_Queue, User_Data

function to_pdu(PQ:PDUqueue) : user_data is
  case PQ in
          to_sp0(ud:user_data)  -> ud
        | q: PDUqueue           -> to_pdu0(q)
  endcase
endfunc

function to_sp(UD:user_data) : PDUqueue is
  case UD in
          to_pdu0(q:PDUqueue)   -> q
        | u: user_data          -> to_sp0(u)
  endcase
endfunc
endmod
======================================================================== *)
(*--------------------------------------------------------------*)
(*      Service Premitive Definition :                          *)
(*      Three types of Service Primitives are necessary.        *)
(*      CCR Service Primitives, ACSE Service Primitives, and    *)
(*      Presentation Layer Service Primitives.                  *)
(*--------------------------------------------------------------*)
(*--------------------------------------------------------------*)
(*      CCR Service Primitives Type :                           *)
(*      This type corresponds the service primitives of CCR.    *)
(*--------------------------------------------------------------*)
type    CCR_Service_Primitive is AE_title,Atomic_Action_Suffix,
                                       Branch_Suffix,
                                       Requestor_Recovery_State,
                                       Responder_Recovery_State,
                                       PDU_Queue
```

```
sorts   CSP
opns
        CBeginReq (*! constructor *) : ae_title (* Atomic Action Identifier Masters Name *),
                    atomic_action_suffix (* Atomic Action Identifier Suffix *),
                    ae_title (* Branch Identifier Superiors Name *),
                    branch_suffix (* Branch Identifier Suffix *),
                    PDUqueue (* User Data *) -> CSP
        CBeginInd (*! constructor *) : ae_title (* Atomic Action Identifier Masters Name *),
                    atomic_action_suffix (* Atomic Action Identifier Suffix *),
                    ae_title (* Branch Identifier Superiors Name *),
                    branch_suffix (* Branch Identifier Suffix *),
                    PDUqueue (* User Data *) -> CSP
        CBeginRsp (*! constructor *) : PDUqueue (* User Data *) -> CSP
        CBeginCnf (*! constructor *) : PDUqueue (* User Data *) -> CSP
        CPrepareReq (*! constructor *) : PDUqueue (* User Data *) -> CSP
        CPrepareInd (*! constructor *) : PDUqueue (* User Data *) -> CSP
        CReadyReq (*! constructor *) : PDUqueue (* User Data *) -> CSP
        CReadyInd (*! constructor *) : PDUqueue (* User Data *) -> CSP
        CCommitReq (*! constructor *) : PDUqueue (* User Data *) -> CSP
        CCommitInd (*! constructor *) : PDUqueue (* User Data *) -> CSP
        CCommitRsp (*! constructor *) : PDUqueue (* User Data *) -> CSP
        CCommitCnf (*! constructor *) : PDUqueue (* User Data *) -> CSP
        CRollbackReq (*! constructor *) : PDUqueue (* User Data *) -> CSP
        CRollbackInd (*! constructor *) : PDUqueue (* User Data *) -> CSP
        CRollbackRsp (*! constructor *) : PDUqueue (* User Data *) -> CSP
        CRollbackCnf (*! constructor *) : PDUqueue (* User Data *) -> CSP
        CRecoverReq (*! constructor *) : requestor_recovery_state,
                      ae_title (* Atomic Action Identifier Masters Name *),
                      atomic_action_suffix (* Atomic Action Identifier
                                              Suffix *),
                      ae_title (* Branch Identifier Superiors Name *),
                      branch_suffix (* Branch Identifier Suffix *),
                      PDUqueue (* User Data *) -> CSP
        CRecoverInd (*! constructor *) : requestor_recovery_state,
                      ae_title (* Atomic Action Identifier Masters Name *),
                      atomic_action_suffix (* Atomic Action Identifier
                                              Suffix *),
                      ae_title (* Branch Identifier Superiors Name *),
                      branch_suffix (* Branch Identifier Suffix *),
                      PDUqueue (* User Data *) -> CSP
        CRecoverRsp (*! constructor *) : responder_recovery_state,
                      ae_title (* Atomic Action Identifier Masters Name *),
                      atomic_action_suffix (* Atomic Action Identifier
                                              Suffix *),
                      ae_title (* Branch Identifier Superiors Name *),
                      branch_suffix (* Branch Identifier Suffix *),
                      PDUqueue (* User Data *) -> CSP
        CRecoverCnf (*! constructor *) : responder_recovery_state,
                      ae_title (* Atomic Action Identifier Masters Name *),
                      atomic_action_suffix (* Atomic Action Identifier
                                              Suffix *),
                      ae_title (* Branch Identifier Superiors Name *),
                      branch_suffix (* Branch Identifier Suffix *),
                      PDUqueue (* User Data *) -> CSP
        CCommitReq_CBeginReq (*! constructor *) : PDUqueue (* User Data *),
                               ae_title (* Atomic Action Identifier Masters
                                           Name *),
                               atomic_action_suffix (* Atomic Action Identifier
                                                       Suffix *),
                               ae_title (* Branch Identifier Superiors Name *),
                               branch_suffix (* Branch Identifier Suffix *),
                               PDUqueue (* User Data *) -> CSP
        CRollbackReq_CBeginReq (*! constructor *) : PDUqueue (* User Data *),
                                 ae_title (* Atomic Action Identifier Masters
                                             Name *),
                                 atomic_action_suffix (* Atomic Action Identifier
                                                         Suffix *),
                                 ae_title (* Branch Identifier Superiors Name *),
                                 branch_suffix (* Branch Identifier Suffix *),
                                 PDUqueue (* User Data *) -> CSP

        CBegin_req,CBegin_ind,CBegin_rsp,CBegin_cnf,CPrepare_req,CPrepare_ind,
        CReady_req,CReady_ind,CCommit_req,CCommit_ind,CCommit_rsp,CCommit_cnf,
        CRollback_req,CRollback_ind,CRollback_rsp,CRollback_cnf,
        CRecover_req,CRecover_ind,CRecover_rsp,CRecover_cnf,
        CCommit_req_CBegin_req,CRollback_req_CBegin_req : -> key

        is_CBeginReq,is_CBeginInd,is_CBeginRsp,is_CBeginCnf,
        is_CPrepareReq,is_CPrepareInd,is_CReadyReq,is_CReadyind,
        is_CCommitReq,is_CCommitInd,is_CCommitRsp,is_CCommitCnf,
        is_CRollbackReq,is_CRollbackInd,is_CRollbackRsp,is_CRollbackCnf,
        is_CRecoverReq,is_CRecoverInd,is_CRecoverRsp,is_CRecoverCnf,
        is_CCommitReq_CBeginReq,is_CRollbackReq_CBeginReq : CSP -> bool

        get_aaimn : CSP -> ae_title (* Atomic Action Identifier Masters Name *)
        get_aais : CSP -> atomic_action_suffix (* Atomic Action Identifier
                                                  Suffix *)
        get_bisn : CSP -> ae_title (* Branch Identifier Superiors Name *)
        get_bis : CSP -> branch_suffix (* Branch Identifier Suffix *)
        get_ud : CSP -> PDUqueue (* User Data *)
        get_udb : CSP -> PDUqueue (* User Data *)
        get_rs : CSP -> requestor_recovery_state
        get_rr : CSP -> responder_recovery_state

        h : CSP -> key
eqns
forall  aaimn:ae_title (* Atomic Action Identifier Masters Name *),
        aais:atomic_action_suffix (* Atomic Actiona Identifier Suffix *),
        bisn:ae_title (* Branch Identifier Superiors Name *),
        bis:branch_suffix (* Branch Identifier Suffix *),
        ud,udb:PDUqueue (* User Data *),
        rs:requestor_recovery_state,rr:responder_recovery_state,csp:CSP
ofsort  key
        CBegin_req = 0;
        CBegin_ind = succ(CBegin_req);
        CBegin_rsp = succ(CBegin_ind);
        CBegin_cnf = succ(CBegin_rsp);
        CPrepare_req = succ(CBegin_cnf);
        CPrepare_ind = succ(CPrepare_req);
        CReady_req = succ(CPrepare_ind);
        CReady_ind = succ(CReady_req);
        CCommit_req = succ(CReady_ind);
        CCommit_ind = succ(CCommit_req);
        CCommit_rsp = succ(CCommit_ind);
        CCommit_cnf = succ(CCommit_rsp);
        CRollback_req = succ(CCommit_cnf);
        CRollback_ind = succ(CRollback_req);
        CRollback_rsp = succ(CRollback_ind);
        CRollback_cnf = succ(CRollback_rsp);
        CRecover_req = succ(CRollback_cnf);
        CRecover_ind = succ(CRecover_req);
        CRecover_rsp = succ(CRecover_ind);
        CRecover_cnf = succ(CRecover_rsp);
        CCommit_req_CBegin_req = succ(CRecover_cnf);
        CRollback_req_CBegin_req = succ(CCommit_req_CBegin_req);

        h(CBeginReq(aaimn,aais,bisn,bis,ud)) = CBegin_req;
        h(CBeginInd(aaimn,aais,bisn,bis,ud)) = CBegin_ind;
        h(CBeginRsp(ud)) = CBegin_rsp;
        h(CBeginCnf(ud)) = CBegin_cnf;
        h(CPrepareReq(ud)) = CPrepare_req;
        h(CPrepareInd(ud)) = CPrepare_ind;
        h(CReadyReq(ud)) = CReady_req;
        h(CReadyInd(ud)) = CReady_ind;
        h(CCommitReq(ud)) = CCommit_req;
        h(CCommitInd(ud)) = CCommit_ind;
        h(CCommitRsp(ud)) = CCommit_rsp;
        h(CCommitCnf(ud)) = CCommit_cnf;
        h(CRollbackReq(ud)) = CRollback_req;
        h(CRollbackInd(ud)) = CRollback_ind;
        h(CRollbackRsp(ud)) = CRollback_rsp;
        h(CRollbackCnf(ud)) = CRollback_cnf;
        h(CRecoverReq(rs,aaimn,aais,bisn,bis,ud)) = CRecover_req;
        h(CRecoverInd(rs,aaimn,aais,bisn,bis,ud)) = CRecover_ind;
        h(CRecoverRsp(rr,aaimn,aais,bisn,bis,ud)) = CRecover_rsp;
        h(CRecoverCnf(rr,aaimn,aais,bisn,bis,ud)) = CRecover_cnf;
        h(CCommitReq_CBeginReq(ud,aaimn,aais,bisn,bis,udb))
                = CCommit_req_CBegin_req;
        h(CRollbackReq_CBeginReq(ud,aaimn,aais,bisn,bis,udb))
                = CRollback_req_CBegin_req ;
ofsort bool
        is_CBeginReq(csp) = h(csp) eq CBegin_req;
        is_CBeginInd(csp) = h(csp) eq CBegin_ind;
        is_CBeginRsp(csp) = h(csp) eq CBegin_rsp;
        is_CBeginCnf(csp) = h(csp) eq CBegin_cnf;
        is_CPrepareReq(csp) = h(csp) eq CPrepare_req;
        is_CPrepareInd(csp) = h(csp) eq CPrepare_ind;
        is_CReadyReq(csp) = h(csp) eq CReady_req;
        is_CReadyInd(csp) = h(csp) eq CReady_ind;
        is_CCommitReq(csp) = h(csp) eq CCommit_req;
        is_CCommitInd(csp) = h(csp) eq CCommit_ind;
        is_CCommitRsp(csp) = h(csp) eq CCommit_rsp;
        is_CCommitCnf(csp) = h(csp) eq CCommit_cnf;
        is_CRollbackReq(csp) = h(csp) eq CRollback_req;
        is_CRollbackInd(csp) = h(csp) eq CRollback_ind;
        is_CRollbackRsp(csp) = h(csp) eq CRollback_rsp;
        is_CRollbackCnf(csp) = h(csp) eq CRollback_cnf;
        is_CRecoverReq(csp) = h(csp) eq CRecover_req;
        is_CRecoverInd(csp) = h(csp) eq CRecover_ind;
        is_CRecoverRsp(csp) = h(csp) eq CRecover_rsp;
        is_CRecoverCnf(csp) = h(csp) eq CRecover_cnf;
        is_CCommitReq_CBeginReq(csp) = h(csp) eq CCommit_req_CBegin_req;
        is_CRollbackReq_CBeginReq(csp) = h(csp) eq CRollback_req_CBegin_req;
ofsort  ae_title (* Atomic Action Identifier Masters Name *)
        get_aaimn(CBeginReq(aaimn,aais,bisn,bis,ud)) = aaimn;
        get_aaimn(CBeginInd(aaimn,aais,bisn,bis,ud)) = aaimn;
        get_aaimn(CRecoverReq(rs,aaimn,aais,bisn,bis,ud)) = aaimn;
        get_aaimn(CRecoverInd(rs,aaimn,aais,bisn,bis,ud)) = aaimn;
        get_aaimn(CRecoverRsp(rr,aaimn,aais,bisn,bis,ud)) = aaimn;
        get_aaimn(CRecoverCnf(rr,aaimn,aais,bisn,bis,ud)) = aaimn;
        get_aaimn(CCommitReq_CBeginReq(ud,aaimn,aais,bisn,bis,udb)) = aaimn;
        get_aaimn(CRollbackReq_CBeginReq(ud,aaimn,aais,bisn,bis,udb)) = aaimn;
ofsort  atomic_action_suffix (* Atomic Action Identifier Suffix *)
        get_aais(CBeginReq(aaimn,aais,bisn,bis,ud)) = aais;
        get_aais(CBeginInd(aaimn,aais,bisn,bis,ud)) = aais;
        get_aais(CRecoverReq(rs,aaimn,aais,bisn,bis,ud)) = aais;
        get_aais(CRecoverInd(rs,aaimn,aais,bisn,bis,ud)) = aais;
        get_aais(CRecoverRsp(rr,aaimn,aais,bisn,bis,ud)) = aais;
        get_aais(CRecoverCnf(rr,aaimn,aais,bisn,bis,ud)) = aais;
        get_aais(CCommitReq_CBeginReq(ud,aaimn,aais,bisn,bis,udb)) = aais;
        get_aais(CRollbackReq_CBeginReq(ud,aaimn,aais,bisn,bis,udb)) = aais;
ofsort  ae_title (* Branch Identifier Superiors Name *)
        get_bisn(CBeginReq(aaimn,aais,bisn,bis,ud)) = bisn;
        get_bisn(CBeginInd(aaimn,aais,bisn,bis,ud)) = bisn;
        get_bisn(CRecoverReq(rs,aaimn,aais,bisn,bis,ud)) = bisn;
        get_bisn(CRecoverInd(rs,aaimn,aais,bisn,bis,ud)) = bisn;
        get_bisn(CRecoverRsp(rr,aaimn,aais,bisn,bis,ud)) = bisn;
        get_bisn(CRecoverCnf(rr,aaimn,aais,bisn,bis,ud)) = bisn;
        get_bisn(CCommitReq_CBeginReq(ud,aaimn,aais,bisn,bis,udb)) = bisn;
        get_bisn(CRollbackReq_CBeginReq(ud,aaimn,aais,bisn,bis,udb)) = bisn;
ofsort  branch_suffix (* Branch Identifier Suffix *)
        get_bis(CBeginReq(aaimn,aais,bisn,bis,ud)) = bis;
        get_bis(CBeginInd(aaimn,aais,bisn,bis,ud)) = bis;
        get_bis(CRecoverReq(rs,aaimn,aais,bisn,bis,ud)) = bis;
        get_bis(CRecoverInd(rs,aaimn,aais,bisn,bis,ud)) = bis;
        get_bis(CRecoverRsp(rr,aaimn,aais,bisn,bis,ud)) = bis;
        get_bis(CRecoverCnf(rr,aaimn,aais,bisn,bis,ud)) = bis;
        get_bis(CCommitReq_CBeginReq(ud,aaimn,aais,bisn,bis,udb)) = bis;
        get_bis(CRollbackReq_CBeginReq(ud,aaimn,aais,bisn,bis,udb)) = bis;
ofsort  PDUqueue (* User Data *)
        get_ud(CBeginReq(aaimn,aais,bisn,bis,ud)) = ud;
        get_ud(CBeginInd(aaimn,aais,bisn,bis,ud)) = ud;
        get_ud(CBeginRsp(ud)) = ud;
        get_ud(CBeginCnf(ud)) = ud;
        get_ud(CPrepareReq(ud)) = ud;
        get_ud(CPrepareInd(ud)) = ud;
        get_ud(CReadyReq(ud)) = ud;
        get_ud(CReadyInd(ud)) = ud;
        get_ud(CCommitReq(ud)) = ud;
        get_ud(CCommitInd(ud)) = ud;
        get_ud(CCommitRsp(ud)) = ud;
        get_ud(CCommitCnf(ud)) = ud;
        get_ud(CRollbackReq(ud)) = ud;
        get_ud(CRollbackInd(ud)) = ud;
        get_ud(CRollbackRsp(ud)) = ud;
        get_ud(CRollbackCnf(ud)) = ud;
        get_ud(CRecoverReq(rs,aaimn,aais,bisn,bis,ud)) = ud;
        get_ud(CRecoverInd(rs,aaimn,aais,bisn,bis,ud)) = ud;
        get_ud(CRecoverRsp(rr,aaimn,aais,bisn,bis,ud)) = ud;
        get_ud(CRecoverCnf(rr,aaimn,aais,bisn,bis,ud)) = ud;
        get_ud(CCommitReq_CBeginReq(ud,aaimn,aais,bisn,bis,udb)) = ud;
        get_ud(CRollbackReq_CBeginReq(ud,aaimn,aais,bisn,bis,udb)) = ud;
        get_udb(CCommitReq_CBeginReq(ud,aaimn,aais,bisn,bis,udb)) = udb;
        get_udb(CRollbackReq_CBeginReq(ud,aaimn,aais,bisn,bis,udb)) = udb;
ofsort  requestor_recovery_state
        get_rs(CRecoverReq(rs,aaimn,aais,bisn,bis,ud)) = rs;
        get_rs(CRecoverInd(rs,aaimn,aais,bisn,bis,ud)) = rs;
ofsort  responder_recovery_state
        get_rr(CRecoverRsp(rr,aaimn,aais,bisn,bis,ud)) = rr;
        get_rr(CRecoverCnf(rr,aaimn,aais,bisn,bis,ud)) = rr;
endtype (* CCR_Service_Primitive *)
(* ---------------------------------------------------------------------
module  CCR_Service_Primitive is AE_title,Atomic_Action_Suffix,
                                Branch_Suffix,
                                Requestor_Recovery_State,
                                Responder_Recovery_State,
                                PDU_Queue
type    CSP is
          CBeginReq (AAIMN: ae_title /* Atomic Action Identifier Masters Name */,
                     AAIS: atomic_action_suffix /* Atomic Action Identifier Suffix */,
                     BISN: ae_title /* Branch Identifier Superiors Name */,
                     BIS: branch_suffix /* Branch Identifier Suffix */,
                     UD:PDUqueue /* User Data */)
        | CBeginInd (AAIMN : ae_title /* Atomic Action Identifier Masters Name */,
                     AAS: atomic_action_suffix /* Atomic Action Identifier Suffix */,
                     BISN: ae_title /* Branch Identifier Superiors Name */,
                     BIS: branch_suffix /* Branch Identifier Suffix */,
                     UD: PDUqueue /* User Data */ )
        | CBeginRsp (UD: PDUqueue /* User Data */)
        | CBeginCnf (UD: PDUqueue /* User Data */)
        | CPrepareReq (UD: PDUqueue /* User Data */)
        | CPrepareInd (UD: PDUqueue /* User Data */)
        | CReadyReq (UD: PDUqueue /* User Data */)
        | CReadyInd (UD: PDUqueue /* User Data */)
        | CCommitReq (UD: PDUqueue /* User Data */)
        | CCommitInd (UD: PDUqueue /* User Data */)
        | CCommitRsp (UD: PDUqueue /* User Data */)
        | CCommitCnf (UD: PDUqueue /* User Data */)
        | CRollbackReq (UD: PDUqueue /* User Data */)
        | CRollbackInd (UD: PDUqueue /* User Data */)
        | CRollbackRsp (UD: PDUqueue /* User Data */)
        | CRollbackCnf (UD: PDUqueue /* User Data */)
        | CRecoverReq (RS: requestor_recovery_state,
                       AAIMN: ae_title /* Atomic Action Identifier Masters Name */,
                       AAIS: atomic_action_suffix /* Atomic Action Identifier Suffix */,
                       BISN: ae_title /* Branch Identifier Superiors Name */,
                       BIS: branch_suffix /* Branch Identifier Suffix */,
                       UD: PDUqueue /* User Data */)
        | CRecoverInd (RS: requestor_recovery_state,
                       AAIMN: ae_title /* Atomic Action Identifier Masters Name */,
                       AAIS: atomic_action_suffix /* Atomic Action Identifier Suffix */,
                       BISN: ae_title /* Branch Identifier Superiors Name */,
                       BIS: branch_suffix /* Branch Identifier Suffix */,
                       UD: PDUqueue /* User Data */)
        | CRecoverRsp (RR: requestor_recovery_state,
                       AAIMN: ae_title /* Atomic Action Identifier Masters Name */,
                       AAIS: atomic_action_suffix /* Atomic Action Identifier Suffix */,
                       BISN: ae_title /* Branch Identifier Superiors Name */,
                       BIS: branch_suffix /* Branch Identifier Suffix */,
                       UD: PDUqueue /* User Data */)
        | CRecoverCnf (RR: requestor_recovery_state,
                       AAIMN: ae_title /* Atomic Action Identifier Masters Name */,
                       AAIS: atomic_action_suffix /* Atomic Action Identifier Suffix */,
                       BISN: ae_title /* Branch Identifier Superiors Name */,
                       BIS: branch_suffix /* Branch Identifier Suffix */,
                       UD: PDUqueue /* User Data */)
        | CCommitReq_CBeginReq (UD: PDUqueue /* User Data */,
                       AAIMN: ae_title /* Atomic Action Identifier Masters Name */,
                       AAIS: atomic_action_suffix /* Atomic Action Identifier Suffix */,
                       BISN: ae_title /* Branch Identifier Superiors Name */,
                       BIS: branch_suffix /* Branch Identifier Suffix */,
                       UDB: PDUqueue /* User Data */)
        | CRollbackReq_CBeginReq (UD: PDUqueue /* User Data */,
                       AAIMN: ae_title /* Atomic Action Identifier Masters Name */,
                       AAIS: atomic_action_suffix /* Atomic Action Identifier Suffix */,
                       BISN: ae_title /* Branch Identifier Superiors Name */,
                       BIS: branch_suffix /* Branch Identifier Suffix */,
                       UDB: PDUqueue /* User Data */)
endtype

/* no translation is need for
        CBegin_req,CBegin_ind,CBegin_rsp,CBegin_cnf,CPrepare_req,CPrepare_ind,
        CReady_req,CReady_ind,CCommit_req,CCommit_ind,CCommit_rsp,CCommit_cnf,
        CRollback_req,CRollback_ind,CRollback_rsp,CRollback_cnf,
        CRecover_req,CRecover_ind,CRecover_rsp,CRecover_cnf,
        CCommit_req_CBegin_req,CRollback_req_CBegin_req : -> key

*/
        succ(CCommit_req_CBegin_req) endfunc

function is_CBeginReq(csp:CSP) : Bool is csp match CBeginReq(...) endfunc

function is_CBeginInd(csp:CSP) : Bool is csp match CBeginInd(...) endfunc
function is_CBeginRsp(csp:CSP) : Bool is csp match CBeginRsp(...) endfunc

function is_CBeginCnf(csp:CSP) : Bool is csp match CBeginCnf(...) endfunc

function is_CPrepareReq(csp:CSP) : Bool is csp match CPrepareReq(...) endfunc

function is_CPrepareInd(csp:CSP) : Bool is csp match CPrepareInd(...) endfunc

function is_CReadyReq(csp:CSP) : Bool is csp match CReadyReq(...) endfunc

function is_CReadyInd(csp:CSP) : Bool is csp match CPrepareInd(...) endfunc

function is_CCommitReq(csp:CSP) : Bool is csp match CCommitReq(...) endfunc

function is_CCommitInd(csp:CSP) : Bool is csp match CCommitInd(...) endfunc

function is_CCommitRsp(csp:CSP) : Bool is csp match CCommitRsp(...) endfunc

function is_CCommitCnf(csp:CSP) : Bool is csp match CCommitCnf(...) endfunc

function is_CRollbackReq(csp:CSP) : Bool is csp match CRollbackReq(...) endfunc

function is_CRollbackInd(csp:CSP) : Bool is csp match CRollbackInd(...) endfunc

function is_CRollbackRsp(csp:CSP) : Bool is csp match CRollbackRsp(...) endfunc

function is_CRollbackCnf(csp:CSP) : Bool is csp match CRollbackCnf(...) endfunc

function is_CRecoverReq(csp:CSP) : Bool is csp match CRecoverReq(...) endfunc

function is_CRecoverInd(csp:CSP) : Bool is csp match CRecoverInd(...) endfunc

function is_CRecoverRsp(csp:CSP) : Bool is csp match CRecoverRsp(...) endfunc

function is_CRecoverCnf(csp:CSP) : Bool is csp match CRecoverCnf(...) endfunc

function is_CCommitReq_CBeginReq(csp:CSP) : Bool is
        csp match CCommitReq_CBeginReq(...)
endfunc

function is_CRollbackReq_CBeginReq(csp:CSP) : Bool is
        csp match CRollbackReq_CBeginReq(...)
endfunc

function get_aaimn(csp:CSP) : ae_title is select AAIMN in csp endfunc

function get_aais(csp:CSP) : atomic_action_suffix is select AAIS in csp endfunc

function get_bisn(csp:CSP) : ae_title is select BISN in csp endfunc

function get_bis(csp:CSP) : branch_suffix is select BIS in csp endfunc

function get_ud(csp:CSP) : PDUqueue is select UD in csp endfunc
function get_udb(csp:CSP) : PDUqueue is select UDB in csp endfunc

function get_rs(csp:CSP) : requestor_recovery_state is select RS in csp endfunc

function get_rr(csp:CSP) : responder_recovery_state is select RR in csp endfunc

/* no translation is needed for function h */
endmod
========================================================================== *)
(*-----------------------------------------------------------*)
(*      ACSE Service Primitives Type                         *)
(*      This Definition does not contain all ACSE Services,  *)
(*      but ones which are necessary for the description of CCR *)
(*      Protocol Machine are only defined.                   *)
(*-----------------------------------------------------------*)
type    ACSE_Service_Primitive is AP_title,AE_Qualifier,
                                Result,Result_Source,Diagnostic,
                                PDU_Queue,Other_Parameters,Boolean,
                                CCR_Service_Primitive
sorts   ASP
opns
        AAssocReq (*! constructor *) : ap_title (* Calling AP Title *),
                    ae_qualifier (* Calling AE Qualifier *),
                    ap_title (* Called AP Title *),
                    ae_qualifier (* Called AE Qualifier *),
                    PDUqueue (* User Data *),
                    other_parameters -> ASP
        AAssocInd (*! constructor *) : ap_title (* Calling AP Title *),
                    ae_qualifier (* Calling AE Qualifier *),
                    ap_title (* Called AP Title *),
                    ae_qualifier (* Called AE Qualifier *),
                    PDUqueue (* User Data *),
                    other_parameters -> ASP
        AAssocRsp (*! constructor *) : ap_title (* Responding AP Title *),
                    ae_qualifier (* Responding AE Qualifier *),
                    result,result_source,diagnostic,
                    PDUqueue (* User Data *),
                    other_parameters -> ASP
        AAssocCnf (*! constructor *) : ap_title (* Responding AP Title *),
                    ae_qualifier (* Responding AE Qualifier *),
                    result,result_source,diagnostic,
                    PDUqueue (* User Data *),
                    other_parameters -> ASP

        AAssoc_req,AAssoc_ind,AAssoc_rsp,AAssoc_cnf : -> key

        is_AAssocReq,is_AAssocInd,is_AAssocRsp,is_AAssocCnf : ASP -> bool

        get_cgat : ASP -> ap_title (* Calling AP Title *)
        get_cgaq : ASP -> ae_qualifier (* Calling AE Qualifier *)
        get_cdat : ASP -> ap_title (* Called AP Title *)
        get_cdaq : ASP -> ae_qualifier (* Called AE Qualifier *)
        get_rat : ASP -> ap_title (* Responding AP Title *)
        get_raq : ASP -> ae_qualifier (* Responding AE Qualifier *)
        get_r : ASP -> result
        get_rs : ASP -> result_source
        get_d : ASP -> diagnostic
        get_ud : ASP -> PDUqueue (* User Data *)
        get_op : ASP -> other_parameters

        h : ASP -> key
eqns
forall  cgat:ap_title (* Calling AP Title *),
        cgaq:ae_qualifier (* Calling AE Qualifier *),
        cdat:ap_title (* Called AP Title *),
        cdaq:ae_qualifier (* Called AE Qualifier *),
        rat:ap_title (* Responding AP Title *),
        raq:ae_qualifier (* Responding AE Qualifier *),
        r:result,rs:result_source,d:diagnostic,
        ud:PDUqueue (* User Data *),
        op:other_parameters,asp:ASP
ofsort  key
        AAssoc_req = succ(CRollback_req_CBegin_req);
        AAssoc_ind = succ(AAssoc_req);
        AAssoc_rsp = succ(AAssoc_ind);
        AAssoc_cnf = succ(AAssoc_rsp);

        h(AAssocReq(cgat,cgaq,cdat,cdaq,ud,op)) = AAssoc_req;
        h(AAssocInd(cgat,cgaq,cdat,cdaq,ud,op)) = AAssoc_ind;
        h(AAssocRsp(rat,raq,r,rs,d,ud,op)) = AAssoc_rsp;
        h(AAssocCnf(rat,raq,r,rs,d,ud,op)) = AAssoc_cnf;
ofsort  bool
        is_AAssocReq(asp) = h(asp) eq AAssoc_req;
        is_AAssocInd(asp) = h(asp) eq AAssoc_ind;
        is_AAssocRsp(asp) = h(asp) eq AAssoc_rsp;
        is_AAssocCnf(asp) = h(asp) eq AAssoc_cnf;
ofsort  ap_title (* Calling AP Title *)
        get_cgat(AAssocReq(cgat,cgaq,cdat,cdaq,ud,op)) = cgat;
        get_cgat(AAssocInd(cgat,cgaq,cdat,cdaq,ud,op)) = cgat;
ofsort  ae_qualifier (* Calling AE Qualifier *)
        get_cgaq(AAssocReq(cgat,cgaq,cdat,cdaq,ud,op)) = cgaq;
        get_cgaq(AAssocInd(cgat,cgaq,cdat,cdaq,ud,op)) = cgaq;
ofsort  ap_title (* Called AP Title *)
        get_cdat(AAssocReq(cgat,cgaq,cdat,cdaq,ud,op)) = cdat;
        get_cdat(AAssocInd(cgat,cgaq,cdat,cdaq,ud,op)) = cdat;
ofsort  ae_qualifier (* Called AE Qualifier *)
        get_cdaq(AAssocReq(cgat,cgaq,cdat,cdaq,ud,op)) = cdaq;
        get_cdaq(AAssocInd(cgat,cgaq,cdat,cdaq,ud,op)) = cdaq;
ofsort  ap_title (* Responding AP Title *)
        get_rat(AAssocRsp(rat,raq,r,rs,d,ud,op)) = rat;
        get_rat(AAssocCnf(rat,raq,r,rs,d,ud,op)) = rat;
ofsort  ae_qualifier (* Responding AE Qualifier *)
        get_raq(AAssocRsp(rat,raq,r,rs,d,ud,op)) = raq;
        get_raq(AAssocCnf(rat,raq,r,rs,d,ud,op)) = raq;
ofsort  result
        get_r(AAssocRsp(rat,raq,r,rs,d,ud,op)) = r;
        get_r(AAssocCnf(rat,raq,r,rs,d,ud,op)) = r;
ofsort  result_source
```

```
        get_rs(AAssocRsp(rat,raq,r,rs,d,ud,op)) = rs;
        get_rs(AAssocCnf(rat,raq,r,rs,d,ud,op)) = rs;
ofsort  diagnostic
        get_d(AAssocRsp(rat,raq,r,rs,d,ud,op)) = d;
        get_d(AAssocCnf(rat,raq,r,rs,d,ud,op)) = d;
ofsort  PDUqueue (* User Data *)
        get_ud(AAssocReq(cgat,cgaq,cdat,cdaq,ud,op)) = ud;
        get_ud(AAssocInd(cgat,cgaq,cdat,cdaq,ud,op)) = ud;
        get_ud(AAssocRsp(rat,raq,r,rs,d,ud,op)) = ud;
        get_ud(AAssocCnf(rat,raq,r,rs,d,ud,op)) = ud;
ofsort  other_parameters
        get_op(AAssocReq(cgat,cgaq,cdat,cdaq,ud,op)) = op;
        get_op(AAssocInd(cgat,cgaq,cdat,cdaq,ud,op)) = op;
        get_op(AAssocRsp(rat,raq,r,rs,d,ud,op)) = op;
        get_op(AAssocCnf(rat,raq,r,rs,d,ud,op)) = op;
endtype (* ACSE_Service_Primitive *)
(* ------------------------------------------------------------------------
module  ACSE_Service_Primitive is AP_title,AE_Qualifier,
                                Result,Result_Source,Diagnostic,
                                PDU_Queue,Other_Parameters,Boolean,
                                CCR_Service_Primitive
type    ASP is
         AAssocReq (CGAT: ap_title /* Calling AP Title */,
                   CGAQ: ae_qualifier /* Calling AE Qualifier */,
                   CDAT: ap_title /* Called AP Title */,
                   CDAQ: ae_qualifier /* Called AE Qualifier */,
                   UD: PDUqueue /* User Data */,
                   OP: other_parameters)
        | AAssocInd (CGAT: ap_title /* Calling AP Title */,
                   CGAQ: ae_qualifier /* Calling AE Qualifier */,
                   CDAT: ap_title /* Called AP Title */,
                   CDAQ: ae_qualifier /* Called AE Qualifier */,
                   UD: PDUqueue /* User Data */,
                   OP: other_parameters)
        | AAssocRsp (RAT: ap_title /* Responding AP Title */,
                    RAQ: ae_qualifier /* Responding AE Qualifier */,
                    R: result,
                    RS: result_source,
                    D: diagnostic,
                    UD: PDUqueue /* User Data */)
                    OP: other_parameters)
        | AAssocCnf (RAT: ap_title /* Responding AP Title */,
                    RAQ: ae_qualifier /* Responding AE Qualifier */,
                    R: result,
                    RS: result_source,
                    D: diagnostic,
                    UD: PDUqueue /* User Data */,
                    OP: other_parameters)
  endcase
endtype

/* no translation is need for AAssoc_req, AAssoc_ind, AAssoc_rsp, AAssoc_cnf */

/* no translation is needed for function h */
```

```
function is_AAssocReq (asp:ASP) : bool is  asp match AAssocReq(...) endfunc

function is_AAssocInd (asp:ASP) : bool is asp match AAssocInd(...) endfunc

function is_AAssocRsp (asp:ASP) : bool is asp match AAssocRsp(...) endfunc

function is_AAssocCnf (asp:ASP) : bool is asp match AAssocCnf(...) endfunc

function get_cgat (asp:ASP) : ap_title is select CGAT in asp endfunc

function get_cgaq (asp:ASP) : ae_qualifier is select CGAQ in asp endfunc

function get_cdat (asp:ASP) : ap_title is select CDAT in asp endfunc

function get_cdaq (asp:ASP) : ae_qualifier is select CDAQ in asp endfunc

function get_rat (asp:ASP) : ap_title is select RAT in asp endfunc

function get_raq (asp:ASP) : ae_qualifier select RAQ in asp endfunc

function get_r (asp:ASP) : result is select R in asp endfunc

function get_rs (asp:ASP) : result_source is select RS in asp endfunc

function get_d (asp:ASP) : diagnostic is select D in asp endfunc

function get_ud (asp:ASP) : PDUqueue is select UD in asp endfunc

function get_op (asp:ASP) : other_parameters is select OP in asp endfunc

endmod
=========================================================================== *)
(*-------------------------------------------------------------*)
(*      Presentation Service Premitive Type :                  *)
(*      This Definition does not contain all Presentation      *)
(*      Services, but ones which are necessary for             *)
(*      the description of CCR Protocol Machine are only       *)
(*      defined.                                               *)
(*-------------------------------------------------------------*)
type    Presentation_Service_Primitive is Sync_Type,Sync_Point_Number,
                                              Resync_Type,Token,
                                              P_Cont_Id_List,Data_Separation,
                                              PDU_Queue,Key,Boolean,
                                              ACSE_Service_Primitive
sorts   PSP
opns
        PConnectInd (*! constructor *) : -> PSP
        PConnectCnf (*! constructor *) : -> PSP

        PSyncMinReq (*! constructor *) : sync_type,sync_point,PDUqueue (* User Data *),
                      data_sep -> PSP
        PSyncMinInd (*! constructor *) : sync_type,sync_point,PDUqueue (* User Data *),
                      data_sep -> PSP
        PSyncMinRsp (*! constructor *) : sync_point,PDUqueue (* User Data *) -> PSP
        PSyncMinCnf (*! constructor *) : sync_point,PDUqueue (* User Data *) -> PSP
        PSyncMajReq (*! constructor *) : sync_point,PDUqueue (* User Data *) -> PSP
        PSyncMajInd (*! constructor *) : sync_point,PDUqueue (* User Data *) -> PSP
        PSyncMajRsp (*! constructor *) : PDUqueue (* User Data *) -> PSP
        PSyncMajCnf (*! constructor *) : PDUqueue (* User Data *) -> PSP
        PReSyncReq (*! constructor *) : resync_type,sync_point,token,
                     PDUqueue (* User Data *) -> PSP
        PReSyncInd (*! constructor *) : resync_type,sync_point,token,p_cont_id_list,
                     PDUqueue (* User Data *) -> PSP
        PReSyncRsp (*! constructor *) : sync_point,token,p_cont_id_list,
                     PDUqueue (* User Data *) -> PSP
        PReSyncCnf (*! constructor *) : sync_point,token,PDUqueue (* User Data *) -> PSP
        PTypedDataReq (*! constructor *) : PDUqueue (* User Data *) -> PSP
        PTypedDataInd (*! constructor *) : PDUqueue (* User Data *) -> PSP

        PConnect_ind,PConnect_cnf,
        PSyncMin_req,PSyncMin_ind,PSyncMin_rsp,PSyncMin_cnf,
        PSyncMaj_req,PSyncMaj_ind,PSyncMaj_rsp,PSyncMaj_cnf,
        PReSync_req,PReSync_ind,PReSync_rsp,PReSync_cnf,
        PTypedData_req,PTypedData_ind : -> key

        is_PConnectInd,is_PConnectCnf,
        is_PSyncMinReq,is_PSyncMinInd,is_PSyncMinRsp,is_PSyncMinCnf,
        is_PSyncMajReq,is_PSyncMajInd,is_PSyncMajRsp,is_PSyncMajCnf,
        is_PReSyncReq,is_PReSyncInd,is_PReSyncRsp,is_PReSyncCnf,
        is_PTypedDataReq,is_PTypedDataInd,is_PSP_for_ACSE : PSP -> Bool

        get_st : PSP -> sync_type
        get_sp : PSP -> sync_point
        get_rt : PSP -> resync_type
        get_t : PSP -> token
        get_pcil : PSP -> p_cont_id_list
        get_ud : PSP -> PDUqueue (* User Data *)

        h : PSP -> key
eqns
forall  st:sync_type,sp:sync_point,rt:resync_type,t:token,pcil:p_cont_id_list,
        ud:PDUqueue (* User Data *),ds:data_sep,psp:PSP
ofsort  key
        PConnect_ind = succ(AAssoc_cnf);
        PConnect_cnf = succ(PConnect_ind);
        PSyncMin_req = succ(PConnect_cnf);
        PSyncMin_ind = succ(PSyncMin_req);
        PSyncMin_rsp = succ(PSyncMin_ind);
        PSyncMin_cnf = succ(PSyncMin_rsp);
        PSyncMaj_req = succ(PSyncMin_cnf);
        PSyncMaj_ind = succ(PSyncMaj_req);
        PSyncMaj_rsp = succ(PSyncMaj_ind);
        PSyncMaj_cnf = succ(PSyncMaj_rsp);
        PReSync_req = succ(PSyncMaj_cnf);
        PReSync_ind = succ(PReSync_req);
        PReSync_rsp = succ(PReSync_ind);
        PReSync_cnf = succ(PReSync_rsp);
        PTypedData_req = succ(PReSync_cnf);
        PTypedData_ind = succ(PTypedData_req);

        h(PConnectInd) = PConnect_ind;
        h(PConnectCnf) = PConnect_cnf;
        h(PSyncMinReq(st,sp,ud,ds)) = PSyncMin_req;
        h(PSyncMinInd(st,sp,ud,ds)) = PSyncMin_ind;
        h(PSyncMinRsp(sp,ud)) = PSyncMin_rsp;
        h(PSyncMinCnf(sp,ud)) = PSyncMin_cnf;
        h(PSyncMajReq(sp,ud)) = PSyncMaj_req;
        h(PSyncMajInd(sp,ud)) = PSyncMaj_ind;
        h(PSyncMajRsp(ud)) = PSyncMaj_rsp;
        h(PSyncMajCnf(ud)) = PSyncMaj_cnf;
        h(PReSyncReq(rt,sp,t,ud)) = PReSync_req;
        h(PReSyncInd(rt,sp,t,pcil,ud)) = PReSync_ind;
        h(PReSyncRsp(sp,t,pcil,ud)) = PReSync_rsp;
        h(PReSyncCnf(sp,t,ud)) = PReSync_cnf;
        h(PTypedDataReq(ud)) = PTypedData_req;
        h(PTypedDataInd(ud)) = PTypedData_ind;
ofsort  Bool
        is_PConnectInd(psp) = h(psp) eq PConnect_ind;
        is_PConnectCnf(psp) = h(psp) eq PConnect_cnf;
        is_PSyncMinReq(psp) = h(psp) eq PSyncMin_req;
        is_PSyncMinInd(psp) = h(psp) eq PSyncMin_ind;
        is_PSyncMinRsp(psp) = h(psp) eq PSyncMin_rsp;
        is_PSyncMinCnf(psp) = h(psp) eq PSyncMin_cnf;
        is_PSyncMajReq(psp) = h(psp) eq PSyncMaj_req;
        is_PSyncMajInd(psp) = h(psp) eq PSyncMaj_ind;
        is_PSyncMajRsp(psp) = h(psp) eq PSyncMaj_rsp;
        is_PSyncMajCnf(psp) = h(psp) eq PSyncMaj_cnf;
        is_PReSyncReq(psp) = h(psp) eq PReSync_req;
        is_PReSyncInd(psp) = h(psp) eq PReSync_ind;
        is_PReSyncRsp(psp) = h(psp) eq PReSync_rsp;
        is_PReSyncCnf(psp) = h(psp) eq PReSync_cnf;
        is_PTypedDataReq(psp) = h(psp) eq PTypedData_req;
        is_PTypedDataInd(psp) = h(psp) eq PTypedData_ind;
        is_PSP_for_ACSE(psp) = is_PConnectInd(psp) or is_PConnectCnf(psp);
ofsort  sync_type
        get_st(PSyncMinReq(st,sp,ud,ds)) = st;
        get_st(PSyncMinInd(st,sp,ud,ds)) = st;
ofsort  sync_point
        get_sp(PSyncMinReq(st,sp,ud,ds)) = sp;
        get_sp(PSyncMinInd(st,sp,ud,ds)) = sp;
        get_sp(PSyncMinRsp(sp,ud)) = sp;
        get_sp(PSyncMinCnf(sp,ud)) = sp;
        get_sp(PSyncMajReq(sp,ud)) = sp;
        get_sp(PSyncMajInd(sp,ud)) = sp;
        get_sp(PReSyncReq(rt,sp,t,ud)) = sp;
        get_sp(PReSyncInd(rt,sp,t,pcil,ud)) = sp;
        get_sp(PReSyncRsp(sp,t,pcil,ud)) = sp;
        get_sp(PReSyncCnf(sp,t,ud)) = sp;
ofsort  resync_type
        get_rt(PReSyncReq(rt,sp,t,ud)) = rt;
        get_rt(PReSyncInd(rt,sp,t,pcil,ud)) = rt;
```

```
ofsort  token
        get_t(PReSyncReq(rt,sp,t,ud)) = t;
        get_t(PReSyncInd(rt,sp,t,pcil,ud)) = t;
        get_t(PReSyncRsp(sp,t,pcil,ud)) = t;
        get_t(PReSyncCnf(sp,t,ud)) = t;
ofsort  p_cont_id_list
        get_pcil(PReSyncInd(rt,sp,t,pcil,ud)) = pcil;
        get_pcil(PReSyncRsp(sp,t,pcil,ud)) = pcil;
ofsort  PDUqueue (* User Data *)
        get_ud(PSyncMinReq(st,sp,ud,ds)) = ud;
        get_ud(PSyncMinInd(st,sp,ud,ds)) = ud;
        get_ud(PSyncMinRsp(sp,ud)) = ud;
        get_ud(PSyncMinCnf(sp,ud)) = ud;
        get_ud(PSyncMajReq(sp,ud)) = ud;
        get_ud(PSyncMajInd(sp,ud)) = ud;
        get_ud(PSyncMajRsp(ud)) = ud;
        get_ud(PSyncMajCnf(ud)) = ud;
        get_ud(PTypedDataReq(ud)) = ud;
        get_ud(PTypedDataInd(ud)) = ud;
        get_ud(PReSyncReq(rt,sp,t,ud)) = ud;
        get_ud(PReSyncInd(rt,sp,t,pcil,ud)) = ud;
        get_ud(PReSyncRsp(sp,t,pcil,ud)) = ud;
        get_ud(PReSyncCnf(sp,t,ud)) = ud;
(*      get_ud(PTypedDataReq(ud)) = ud;
        get_ud(PTypedDataInd(ud)) = ud; *)
endtype (* Presentation_Service_Primitive *)
(* ---------------------------------------------------------------------
module  Presentation_Service_Primitive is Sync_Type,Sync_Point_Number,
                                        Resync_Type,Token,
                                        P_Cont_Id_List,Data_Separation,
                                        PDU_Queue,Key,Boolean,
                                        ACSE_Service_Primitive

type    PSP     is
          PConnectInd
        | PConnectCnf
        | PSyncMinReq (ST: sync_type,
                SP: sync_point,
                UD: PDUqueue,
                DS: data_sep)
        | PSyncMinInd (ST: sync_type,
                SP: sync_point,
                UD: PDUqueue,
                DS: data_sep)
        | PSyncMinRsp (SP: sync_point,UD: PDUqueue)
        | PSyncMinCnf (SP: sync_point,UD: PDUqueue)
        | PSyncMajReq (SP: sync_point,UD: PDUqueue)
        | PSyncMajInd (SP: sync_point,UD: PDUqueue)
        | PSyncMajRsp (UD: PDUqueue)
        | PSyncMajCnf (UD: PDUqueue)
        | PReSyncReq (RST: resync_type,
                SP: sync_point,
                T: token,
                UD: PDUqueue)
        | PReSyncInd (RST: resync_type,
                SP: sync_point,
                T: token,
                PCIL: p_cont_id_list,
                UD: PDUqueue)
        | PReSyncRsp (SP: sync_point,
                T: token,
                PCIL: p_cont_id_list,
                UD: PDUqueue)
        | PReSyncCnf  (SP: sync_point,
                T: token,
                UD: PDUqueue)
        | PTypedDataReq (UD: PDUqueue)
        | PTypedDataInd (UD: PDUqueue)
endtype

/*  no translation is needed for PConnect_ind, PConnect_cnf, PSyncMin_req,
        PSyncMin_ind, PSyncMin_rsp, PSyncMin_cnf, PSyncMaj_req, PSyncMaj_ind,
        PSyncMaj_rsp, PSyncMaj_cnf, PReSync_req, PReSync_ind, PReSync_rsp,
        PReSync_cnf, PTypedData_req, PTypedData_ind
*/
/* no translation is needed for function h */

function is_PConnectInd(psp:PSP) : Bool is psp match PConnectInd endfunc

function is_PConnectCnf(psp:PSP) : Bool is psp match PConnectCnf endfunc

function is_PSyncMinReq(psp:PSP) : Bool is psp match PSyncMinReq(...) endfunc

function is_PSyncMinInd(psp:PSP) : Bool is psp match PSyncMinInd(...) endfunc

function is_PSyncMinRsp(psp:PSP) : Bool is psp match PSyncMinRsp(...) endfunc

function is_PSyncMinCnf(psp:PSP) : Bool is psp match PSyncMinCnf(...) endfunc

function is_PSyncMajReq(psp:PSP) : Bool is psp match PSyncMajReq(...) endfunc

function is_PSyncMajInd(psp:PSP) : Bool is psp match PSyncMajInd(...) endfunc

function is_PSyncMajRsp(psp:PSP) : Bool is psp match PSyncMajRsp(...) endfunc

function is_PSyncMajCnf(psp:PSP) : Bool is psp match PSyncMajCnf(...) endfunc

function is_PReSyncReq(psp:PSP) : Bool is psp match PReSyncReq(...) endfunc

function is_PReSyncInd(psp:PSP) : Bool is psp match PReSyncInd(...) endfunc

function is_PReSyncRsp(psp:PSP) : Bool is psp match PReSyncRsp(...) endfunc

function is_PReSyncCnf(psp:PSP) : Bool is psp match PReSyncCnf(...) endfunc

function is_PTypedDataReq(psp:PSP) : Bool is psp match PTypedDataReq(...) endfunc

function is_PTypedDataInd(psp:PSP) : Bool is psp match PTypedDataInd(...) endfunc
function is_PSP_for_ACSE(psp:PSP) : Bool is psp match  endfunc

function get_st (psp: PSP) : sync_type is select ST in psp endfunc

function get_sp (psp: PSP) : sync_point is select SP in psp endfunc

function get_rt (psp: PSP) : resync_type is select RST in psp endfunc

function get_t (psp: PSP) : token is select T in psp endfunc

function get_pcil (psp: PSP) : p_cont_id_liST is select PCIL in psp endfunc

function get_ud (psp: PSP) : PDUqueue is select UD in psp endfunc
endmod
=========================================================================== *)
(*------------------------------------------------------------*)
(*      General Service Primitive Type :                       *)
(*      This is a general service primitive type which         *)
(*      represents Service Primitives in Application Layer and  *)
(*      Presentation Layer. The type for general service       *)
(*      primitivies is constructed from the Service Primitive   *)
(*      types of ASEs which are CCR and ACSE and that of       *)
(*      Presentation Layer.                                    *)
(*------------------------------------------------------------*)
type    Service_Primitive is CCR_Service_Primitive,
                            ACSE_Service_Primitive,
                            Presentation_Service_Primitive
sorts   SP
opns
        sp (*! constructor *) : CSP -> SP
        sp (*! constructor *) : ASP -> SP
        sp (*! constructor *) : PSP -> SP
        is_ccr_sp : SP -> Bool
        is_acse_sp : SP -> Bool
        is_pl_sp : SP -> Bool
        get_csp : SP -> CSP
        get_asp : SP -> ASP
        get_psp : SP -> PSP
eqns
forall  c:CSP,a:ASP,p:PSP
ofsort  CSP
        get_csp(sp(c)) = c;
ofsort  ASP
        get_asp(sp(a)) = a;
ofsort  PSP
        get_psp(sp(p)) = p;
ofsort  Bool
        is_ccr_sp(sp(c)) = true;
        is_ccr_sp(sp(a)) = false;
        is_ccr_sp(sp(p)) = false;

        is_acse_sp(sp(c)) = false;
        is_acse_sp(sp(a)) = true;
        is_acse_sp(sp(p)) = false;

        is_pl_sp(sp(c)) = false;
        is_pl_sp(sp(a)) = false;
        is_pl_sp(sp(p)) = true;
endtype (* Service_Primitive *)
(* ---------------------------------------------------------------------------
module  Service_Primitive is  CCR_Service_Primitive,
                              ACSE_Service_Primitive,
                              Presentation_Service_Primitive
type    SP is
          sp (csp: CSP)
        | sp (asp: ASP)
        | sp (psp: PSP)
endtype

function is_ccr_sp (sp:SP) : Bool is sp match sp(any CSP) endfunc
function is_acse_sp (sp:SP) : Bool is sp match sp(any ASP) endfunc
function is_pl_sp (sp:SP) : Bool is sp match sp(any PSP) endfunc

function get_csp (sp:SP) : CSP is select csp in sp endfunc
function get_asp (sp:SP) : ASP is select asp in sp endfunc
function get_psp (sp:SP) : PSP is select psp in sp endfunc
endmod
========================================================================== *)
(*-------------------------------------------------------------*)
(*      PDU Type :                                             *)
(*      In this description, two PDU types are defined.        *)
(*      The first is CCR PDU and the second is other ASE PDU.  *)
(*      The former is semantically impotant and the latter is  *)
(*      a just place holder. But these PDUs are necessary      *)
(*      in the discussion of concatenation/deconcatenation of  *)
(*      PDUs.                                                  *)
(*-------------------------------------------------------------*)
(*-------------------------------------------------------------*)
(*      CCR PDU Type                                           *)
(*-------------------------------------------------------------*)
type    CCR_PDU is CCR_Version_Number,Atomic_Action_Identifier,
                   Branch_Identifier,Branch_Suffix,
                   Requestor_Recovery_State,Responder_Recovery_State,
                   User_Data,Key
sorts   CPDU
opns    c_initialize_ri (*! constructor *) : ccr_ver -> CPDU
        c_initialize_rc (*! constructor *) : ccr_ver -> CPDU
        c_begin_ri (*! constructor *) : atomic_action_identifier,
                     branch_suffix,
                     user_data -> CPDU
        c_begin_rc (*! constructor *) : user_data -> CPDU
        c_prepare_ri (*! constructor *) : user_data -> CPDU
        c_ready_ri (*! constructor *) : user_data -> CPDU
        c_commit_ri (*! constructor *) : user_data -> CPDU
        c_commit_rc (*! constructor *) : user_data -> CPDU
        c_rollback_ri (*! constructor *) : user_data -> CPDU
        c_rollback_rc (*! constructor *) : user_data -> CPDU
        c_recover_ri (*! constructor *) : atomic_action_identifier,
                       branch_identifier,
                       requestor_recovery_state,
                       user_data -> CPDU
        c_recover_rc (*! constructor *) : atomic_action_identifier,
                       branch_identifier,
                       responder_recovery_state,
                       user_data -> CPDU

        c_empty (*! constructor *) : -> CPDU

        c_initialize_ri,c_initialize_rc,c_begin_ri,c_begin_rc,
        c_prepare_ri,c_ready_ri,c_commit_ri,c_commit_rc,
        c_rollback_ri,c_rollback_rc,c_recover_ri,c_recover_rc : -> key

        is_c_initialize_ri,is_c_initialize_rc,
        is_c_begin_ri,is_c_begin_rc,is_c_prepare_ri,is_c_ready_ri,
        is_c_commit_ri,is_c_commit_rc,is_c_rollback_ri,is_c_rollback_rc,
        is_c_recover_ri,is_c_recover_rc : CPDU -> Bool

        get_v : CPDU -> ccr_ver
        get_aai : CPDU -> atomic_action_identifier
        get_bi : CPDU -> branch_identifier
        get_bs : CPDU -> branch_suffix
        get_ud : CPDU -> user_data
        get_rs : CPDU -> requestor_recovery_state
        get_rr : CPDU -> responder_recovery_state

        h : CPDU -> key
eqns
forall  v:ccr_ver,aai:atomic_action_identifier,bs:branch_suffix,
        bi:branch_identifier,ud,udb:user_data,rs:requestor_recovery_state,
        rr:responder_recovery_state,cpdu:CPDU
ofsort  key
        c_initialize_ri = 0;
        c_initialize_rc = succ(c_initialize_ri);
        c_begin_ri = succ(c_initialize_rc);
        c_begin_rc = succ(c_begin_ri);
        c_prepare_ri = succ(c_begin_rc);
        c_ready_ri = succ(c_prepare_ri);
        c_commit_ri = succ(c_ready_ri);
        c_commit_rc = succ(c_commit_ri);
        c_rollback_ri = succ(c_commit_rc);
        c_rollback_rc = succ(c_rollback_ri);
        c_recover_ri = succ(c_rollback_rc);
        c_recover_rc = succ(c_recover_ri);

        h(c_initialize_ri(v)) = c_initialize_ri;
        h(c_initialize_rc(v)) = c_initialize_rc;
        h(c_begin_ri(aai,bs,ud)) = c_begin_ri;
        h(c_begin_rc(ud)) = c_begin_rc;
        h(c_prepare_ri(ud)) = c_prepare_ri;
        h(c_ready_ri(ud)) = c_ready_ri;
        h(c_commit_ri(ud)) = c_commit_ri;
        h(c_commit_rc(ud)) = c_commit_rc;
        h(c_rollback_ri(ud)) = c_rollback_ri;
        h(c_rollback_rc(ud)) = c_rollback_rc;
        h(c_recover_ri(aai,bi,rs,ud)) = c_recover_ri;
        h(c_recover_rc(aai,bi,rr,ud)) = c_recover_rc;
ofsort  bool
        is_c_initialize_ri(cpdu) = h(cpdu) eq c_initialize_ri;
        is_c_initialize_rc(cpdu) = h(cpdu) eq c_initialize_rc;
        is_c_begin_ri(cpdu) = h(cpdu) eq c_begin_ri;
        is_c_begin_rc(cpdu) = h(cpdu) eq c_begin_rc;
        is_c_prepare_ri(cpdu) = h(cpdu) eq c_prepare_ri;
        is_c_ready_ri(cpdu) = h(cpdu) eq c_ready_ri;
        is_c_commit_ri(cpdu) = h(cpdu) eq c_commit_ri;
        is_c_commit_rc(cpdu) = h(cpdu) eq c_commit_rc;
        is_c_rollback_ri(cpdu) = h(cpdu) eq c_rollback_ri;
        is_c_rollback_rc(cpdu) = h(cpdu) eq c_rollback_rc;
        is_c_recover_ri(cpdu) = h(cpdu) eq c_recover_ri;
        is_c_recover_rc(cpdu) = h(cpdu) eq c_recover_rc;
ofsort  ccr_ver
        get_v(c_initialize_ri(v)) = v;
        get_v(c_initialize_rc(v)) = v;
ofsort  atomic_action_identifier
        get_aai(c_begin_ri(aai,bs,ud)) = aai;
        get_aai(c_recover_ri(aai,bi,rs,ud)) = aai;
        get_aai(c_recover_rc(aai,bi,rr,ud)) = aai;
ofsort  branch_suffix
        get_bs(c_begin_ri(aai,bs,ud)) = bs;
ofsort  branch_identifier
        get_bi(c_recover_ri(aai,bi,rs,ud)) = bi;
        get_bi(c_recover_rc(aai,bi,rr,ud)) = bi;
ofsort  user_data
        get_ud(c_begin_ri(aai,bs,ud)) = ud;
        get_ud(c_begin_rc(ud)) = ud;
        get_ud(c_prepare_ri(ud)) = ud;
        get_ud(c_ready_ri(ud)) = ud;
        get_ud(c_commit_ri(ud)) = ud;
        get_ud(c_commit_rc(ud)) = ud;
        get_ud(c_rollback_ri(ud)) = ud;
        get_ud(c_rollback_rc(ud)) = ud;
        get_ud(c_recover_ri(aai,bi,rs,ud)) = ud;
        get_ud(c_recover_rc(aai,bi,rr,ud)) = ud;
ofsort  requestor_recovery_state
        get_rs(c_recover_ri(aai,bi,rs,ud)) = rs;
ofsort  responder_recovery_state
        get_rr(c_recover_rc(aai,bi,rr,ud)) = rr;
endtype (* CCR_PDU *)
(* ------------------------------------------------------------------------
module  CCR_PDU is  CCR_Version_Number,Atomic_Action_Identifier,
                    Branch_Identifier,Branch_Suffix,
                    Requestor_Recovery_State,Responder_Recovery_State,
                    User_Data,Key
type    CPDU is
          c_initialize_ri (V: ccr_ver)
        | c_initialize_rc (V: ccr_ver)
        | c_begin_ri (AAI: atomic_action_identifier,
                      BS: branch_suffix,
                      UD: user_data)
        | c_begin_rc (UD: user_data)
        | c_prepare_ri (UD: user_data)
        | c_ready_ri (UD: user_data)
        | c_commit_ri (UD: user_data)
        | c_commit_rc (UD: user_data)
        | c_rollback_ri (UD: user_data)
        | c_rollback_rc (UD: user_data)
        | c_recover_ri (AAI: atomic_action_identifier,
                        BI: branch_identifier,
                        RS: requestor_recovery_state,
                        UD: user_data)
        | c_recover_rc (AAI: atomic_action_identifier,
                        BI: branch_identifier,
                        RR: responder_recovery_state,
                        UD: user_data)
        | c_empty
endtype

/* no translation is needed for
        c_initialize_ri,c_initialize_rc,c_begin_ri,c_begin_rc,
        c_prepare_ri,c_ready_ri,c_commit_ri,c_commit_rc,
        c_rollback_ri,c_rollback_rc,c_recover_ri,c_recover_rc : -> key
*/

/* no translation is needed for function h */

function is_c_initialize_ri(cpdu: CPDU): Bool is cpdu match c_initialize_ri(...) endfunc

function is_c_initialize_rc(cpdu: CPDU): Bool is cpdu match c_initialize_rc(...) endfunc

function is_c_begin_ri(cpdu: CPDU): Bool is cpdu match c_begin_ri(...) endfunc

function is_c_begin_rc(cpdu: CPDU): Bool is cpdu match c_begin_rc(...) endfunc

function is_c_prepare_ri(cpdu: CPDU): Bool is cpdu match c_prepare_ri(...) endfunc

function is_c_ready_ri(cpdu: CPDU): Bool is cpdu match c_ready_ri(...) endfunc

function is_c_commit_ri(cpdu: CPDU): Bool is cpdu match c_commit_ri(...) endfunc

function is_c_commit_rc(cpdu: CPDU): Bool iscpdu match c_commit_rc(...) endfunc

function is_c_rollback_ri(cpdu: CPDU): Bool is cpdu match c_rollback_ri(...) endfunc

function is_c_rollback_rc(cpdu: CPDU): Bool is cpdu match c_rollback_rc(...) endfunc

function is_c_recover_ri(cpdu: CPDU): Bool is cpdu match c_recover_ri(...) endfunc

function is_c_recover_rc(cpdu: CPDU): Bool is cpdu match c_recover_rc(...) endfunc

function get_v (cpdu: CPDU) : ccr_ver is select V in cpdu endfunc
function get_aai (cpdu: CPDU) : atomic_action_identifier is select AAI in cpdu endfunc

function get_bi (cpdu: CPDU) : branch_identifier is select BS in cpdu endfunc

function get_bs (cpdu: CPDU) : branch_suffix is select BI in cpdu endfunc

function get_ud (cpdu: CPDU) : user_data is select UD in cpdu endfunc

function get_rs (cpdu: CPDU) : requestor_recovery_state is select RS in cpdu endfunc

function get_rr (cpdu: CPDU) : responder_recovery_state is select RR in cpdu endfunc

endmod
========================================================================== *)
(*-------------------------------------------------------------*)
(*      Application Protocol Data Unit :                       *)
(*      This type represents the general Application Layer PDUs.*)
(*      Their structure are not important in this description.  *)
(*      Therefore, a just place holder is defined.             *)
(*-------------------------------------------------------------*)
type    Sub_A_PDU is General renamedby
sortnames
        APDU for general
opnnames
        a_empty for null
endtype (* Sub_A_PDU *)

type    A_PDU is Sub_A_PDU,Key,CCR_PDU
opns
        h : APDU -> key
        apdu : -> key
eqns
forall  a:APDU
ofsort  key
        h(a) = apdu;
        apdu = succ(c_recover_rc);
endtype (* A_PDU *)
(* -------------------------------------------------------------------------
module  Sub_A_PDU is General{sorts APDU for general
                        opns a_empty for null}
endmod

module  A_PDU is  Sub_A_PDU,Key,CCR_PDU
/* no translation is needed for function apdu */
/* no translation is needed for function h */
endmod
========================================================================== *)
(*-------------------------------------------------------------*)
(*      Transformation from CCR Service Primitive to CCR PDU :  *)
(*      The aim of this type is to define two transformation   *)
(*      functions, "begin_trans_sp_to_pdu" and                 *)
(*      "recover_trans_sp_to_pdu".                             *)
(*      "begin_trans_sp_to_pdu" transforms as follows;         *)
(*              C-Begin request -> c_begin_ri                *)
(*              C-Commit request + C-Begin request           *)
(*                              -> c-begin-ri                *)
(*              C-Rollback request + C-Begin request         *)
(*                              -> c-begin-ri.               *)
(*      "recover_trans_sp_to_pdu" transforms as follows;     *)
(*              C-Recover request -> c_recover_ri            *)
(*              C-Recover response -> c-recover-rc           *)
(*      The transformation of C-Commit request and C-Rollback  *)
(*      request service primitices and their concatenation with *)
(*      c-begin-ri are described in the process part.        *)
(*      The important point is the transformations of        *)
(*      "sender/receiver" in CCR version 2. The use of "sender/ *)
(*      receiver" is a sender's option. This is represented by  *)
(*      non-deterministic expressions of equations whose     *)
(*      premiss are not exclusive.                           *)
(*-----------------------------------------------------------*)
type    CCR_Service_Primitive_To_CCR_PDU is CCR_PDU,CCR_Service_Primitive,
                                            AE_Title,
                                            Transformation_User_Data
opns
        begin_trans_sp_to_pdu : CSP,ccr_ver,ae_title,ae_title -> CPDU
        recover_trans_sp_to_pdu : CSP,ccr_ver,ae_title,ae_title -> CPDU
eqns
forall  csp:CSP,ver:ccr_ver,aet1,aet2:ae_title
ofsort  CPDU
        is_CBeginReq(csp) or is_CCommitReq_CBeginReq(csp)
                                or is_CRollbackReq_CBeginReq(csp) =>
                begin_trans_sp_to_pdu(csp,ver,aet1,aet2)
                        = c_begin_ri(
                                atomic_action_identifier(
                                        masters_name(get_aaimn(csp)),
                                        get_aais(csp)),
                                get_bis(csp),
                                to_pdu(get_ud(csp)));

        (is_CBeginReq(csp) or is_CCommitReq_CBeginReq(csp)
                                or is_CRollbackReq_CBeginReq(csp))
                        and (get_aaimn(csp) eq aet1) =>
                begin_trans_sp_to_pdu(csp,ver2,aet1,aet2)
                        = c_begin_ri(
                                atomic_action_identifier(
                                        masters_name(sender),
                                        get_aais(csp)),
                                get_bis(csp),
                                to_pdu(get_ud(csp)));

        is_CRecoverReq(csp) =>
                recover_trans_sp_to_pdu(csp,ver,aet1,aet2)
                        = c_recover_ri(
                                atomic_action_identifier(
                                        masters_name(get_aaimn(csp)),
                                        get_aais(csp)),
                                branch_identifier(
                                superiors_name(get_bisn(csp)),
                                get_bis(csp)),
                        get_rs(csp),
                        to_pdu(get_ud(csp)));

        is_CRecoverReq(csp) and (get_aaimn(csp) eq aet1)
                                and (get_bisn(csp) eq aet1) =>
                recover_trans_sp_to_pdu(csp,ver2,aet1,aet2)
                        = c_recover_ri(
                                atomic_action_identifier(
                                        masters_name(sender),
                                        get_aais(csp)),
                                branch_identifier(
                                        superiors_name(sender),
                                        get_bis(csp)),
                                get_rs(csp),
                                to_pdu(get_ud(csp)));

        is_CRecoverReq(csp) and (get_aaimn(csp) eq aet2)
                                and (get_bisn(csp) eq aet2) =>
                recover_trans_sp_to_pdu(csp,ver2,aet1,aet2)
                        = c_recover_ri(
                                atomic_action_identifier(
                                        masters_name(receiver),
                                        get_aais(csp)),
                                branch_identifier(
                                        superiors_name(receiver),
                                        get_bis(csp)),
                                get_rs(csp),
                                to_pdu(get_ud(csp)));

        is_CRecoverRsp(csp) =>
                recover_trans_sp_to_pdu(csp,ver,aet1,aet2)
                        = c_recover_rc(
                                atomic_action_identifier(
                                        masters_name(get_aaimn(csp)),
                                        get_aais(csp)),
                                branch_identifier(
                                        superiors_name(get_bisn(csp)),
                                        get_bis(csp)),
                                get_rr(csp),
                                to_pdu(get_ud(csp)));

        is_CRecoverRsp(csp) and (get_aaimn(csp) eq aet1)
                                and (get_bisn(csp) eq aet1) =>
                recover_trans_sp_to_pdu(csp,ver2,aet1,aet2)
                        = c_recover_rc(
                                atomic_action_identifier(
                                        masters_name(sender),
                                        get_aais(csp)),
                                branch_identifier(
                                        superiors_name(sender),
                                        get_bis(csp)),
                                get_rr(csp),
                                to_pdu(get_ud(csp)));

        is_CRecoverRsp(csp) and (get_aaimn(csp) eq aet2)
                                        and (get_bisn(csp) eq aet2) =>
                recover_trans_sp_to_pdu(csp,ver2,aet1,aet2)
                        = c_recover_rc(
                                atomic_action_identifier(
                                        masters_name(receiver),
                                        get_aais(csp)),
                                branch_identifier(
                                        superiors_name(receiver),
                                        get_bis(csp)),
                                get_rr(csp),
                                to_pdu(get_ud(csp)));
endtype (* CCR_Service_Primitive_To_CCR_PDU *)
(* ---------------------------------------------------------------------------
module  CCR_Service_Primitive_To_CCR_PDU is  CCR_PDU,CCR_Service_Primitive,
                                                AE_Title,
                                                Transformation_User_Data

function begin_trans_sp_to_pdu (csp: CSP,ver: ccr_ver,
                at1: ae_title,aet2: ae_title) : CPDU is
  case csp in
      CBeginReq(...)
    | CCommitReq_CBeginReq(...)
    | CRollbackReq_CBeginReq(...) ->
        c_begin_ri(atomic_action_identifier(masters_name(select AAIMN in csp),
                select AAIS in csp), select BIS in csp, to_pdu(select UD in csp))
    | CBeginReq(...)
    | CCommitReq_CBeginReq(...)
    | CRollbackReq_CBeginReq(...) when (select AAIMN in csp) eq aet1 ->
        c_begin_ri(atomic_action_identifier(masters_name(sender),  select aais in csp),
                select BIS in csp, to_pdu(select UD in csp))
endfunc

function recover_trans_sp_to_pdu  (csp: CSP,ver: ccr_ver,
                        at1: ae_title,aet2: ae_title) : CPDU is
  case csp in
          CRecoverReq(AAIMN:=aet1,BISN:=aet1,...) ->
                c_recover_ri(atomic_action_identifier(masters_name(sender),
                        select AAIS in csp), branch_identifier(superiors_name(sender),
                        select BIS in csp), select RS in csp, to_pdu(select UD in csp))
        | CRecoverReq(AAIMN:=aet2,BISN:=aet2,...) ->
                c_recover_ri(atomic_action_identifier(masters_name(receiver),
                        select AAIS in csp), branch_identifier(superiors_name(receiver),
                        select BIS in csp), select RS in csp, to_pdu(select UD in csp))
        | CRecoverReq(...) ->
                c_recover_ri(atomic_action_identifier(masters_name(select AAIM in csp),
                        select AAIS in csp), branch_identifier(superiors_name(
                        select BISN in csp), select BIS in csp), select RS in csp,
                        to_pdu(select UD in csp))
        | CRecoverRsp(AAIMN:=aet1,BISN:=aet1,...) ->
                c_recover_rc(atomic_action_identifier(masters_name(sender),
                        select AAIS in csp), branch_identifier(superiors_name(sender),
                        select BIS in csp), select RR in csp, to_pdu(select UD in csp))
        | CRecoverRsp(AAIMN:=aet2,BISN:=aet2,...) ->
                c_recover_rc(atomic_action_identifier(masters_name(receiver),
                        select AAIS in csp), branch_identifier(superiors_name(receiver),
                        select BIS in csp), select RR in csp, to_pdu(select UD in csp))
        | CRecoverRsp(...) ->
                c_recover_rc(atomic_action_identifier(masters_name(select AAIM in csp),
                        select AAIS in csp), branch_identifier(superiors_name(select
                        BISN in csp), select BIS in csp), select RR in csp, to_pdu(select
                        UD in csp))
  endcase
endfunc
endmod
=========================================================================== *)
(*---------------------------------------------------------------*)
(*      Transformation from CCR PDU to CCR Service Primitive :  *)
(*      The aim of this type is similar to that of              *)
(*      CCR_Service_Primitive_To_CCR_PDU.                       *)
(*      CCR_Service_Primitive_To_CCR_PDU includes the functions *)
(*      which transform CCR service primitives into CCR PDUs.   *)
(*      In CCR_PDU_To_CCR_Service_Primitives, the functions     *)
(*      which transform CCR PDUs into CCR service primitives    *)
(*      are defined.                                            *)
(*      The important point of this type is the same as that of *)
(*      the previous type, that is, the treatment of            *)
(*      "sender/receiver". By the facility of "get_aet" this is *)
(*      treated.                                                *)
(*---------------------------------------------------------------*)
type    CCR_PDU_To_CCR_Service_Primitive is CCR_PDU,CCR_Service_Primitive,
                                                AE_Title,
                                                Transformation_User_Data
opns
        begin_trans_pdu_to_sp : CPDU,ccr_ver,ae_title,ae_title -> CSP
        recover_trans_pdu_to_sp : CPDU,ccr_ver,ae_title,ae_title -> CSP
        get_aaimn,
        get_bisn : CPDU,ccr_ver,ae_title,ae_title -> ae_title
        get_aais : CPDU -> atomic_action_suffix
        get_bis :  CPDU -> branch_suffix
eqns
forall  pdu:CPDU,ver:ccr_ver,aet1,aet2:ae_title
ofsort  ae_title
is_c_begin_ri(pdu) or is_c_recover_ri(pdu) or is_c_recover_rc(pdu) =>
        get_aaimn(pdu,ver,aet1,aet2) =
                get_aet(get_mn(get_aai(pdu)),ver,aet1,aet2);
is_c_begin_ri(pdu) =>
        get_bisn(pdu,ver,aet1,aet2) = aet2;     (* sender *)
is_c_recover_ri(pdu) or is_c_recover_rc(pdu) =>
        get_bisn(pdu,ver,aet1,aet2) =
                get_aet(get_sn(get_bi(pdu)),ver,aet1,aet2);
ofsort  atomic_action_suffix
is_c_begin_ri(pdu) or is_c_recover_ri(pdu) or is_c_recover_rc(pdu) =>
        get_aais(pdu)   = get_aas(get_aai(pdu));
ofsort  branch_suffix
is_c_begin_ri(pdu) =>
```

```
        get_bis(pdu)    = get_bs(pdu);
is_c_recover_ri(pdu) or is_c_recover_rc(pdu) =>
        get_bis(pdu)    = get_bs(get_bi(pdu));
ofsort  CSP
        begin_trans_pdu_to_sp(pdu,ver,aet1,aet2)
                = CBeginInd(get_aaimn(pdu,ver,aet1,aet2),
                            get_aais(pdu),
                            get_bisn(pdu,ver,aet1,aet2),
                            get_bis(pdu),
                            to_sp(get_ud(pdu)));

        is_c_recover_ri(pdu) =>
                recover_trans_pdu_to_sp(pdu,ver,aet1,aet2)
                        = CRecoverInd(get_rs(pdu),
                                  get_aaimn(pdu,ver,aet1,aet2),
                                  get_aais(pdu),
                                  get_bisn(pdu,ver,aet1,aet2),
                                  get_bis(pdu),
                                  to_sp(get_ud(pdu)));

        is_c_recover_rc(pdu) =>
                recover_trans_pdu_to_sp(pdu,ver,aet1,aet2)
                        = CRecoverCnf(get_rr(pdu),
                                  get_aaimn(pdu,ver,aet1,aet2),
                                  get_aais(pdu),
                                  get_bisn(pdu,ver,aet1,aet2),
                                  get_bis(pdu),
                                  to_sp(get_ud(pdu)));
endtype (* CCR_PDU_To_CCR_Service_Primitive *)
(* ------------------------------------------------------------------------
module  CCR_PDU_To_CCR_Service_Primitive is  CCR_PDU,CCR_Service_Primitive,
                                                AE_Title,
                                                Transformation_User_Data

function get_aaimn (cpdu: CPDU,ver: ccr_ver,
        aet1: ae_title,aet2: ae_title) : ae_title is
  case pdu in
          c_begin_ri(...)
        | c_recover_ri(...)
        | c_recover_rc(...) -> select AET in (select MN in (select AAI in pdu))
  endcase
endfunc

function get_bisn (cpdu: CPDU,ver: ccr_ver,
                aet1: ae_title,aet2: ae_title) : ae_title is
  case pdu in
          c_begin_ri(...) -> aet2
        | c_recover_ri(...)
        | c_recover_rc(...) ->  select AET in (select SN in (select BI in pdu))
  endcase
endfunc

function get_aais (pdu: CPDU) : atomic_action_suffix is
  case pdu in
```

```
          c_begin_ri(...)
        | c_recover_ri(...)
        | c_recover_rc(...) -> select AAS in (select AAI in pdu))
  endcase
endfunc

function get_bis (pdu: CPDU) : branch_suffix is
  case pdu in
          c_begin_ri(...) -> select BS in pdu /* more PM */
        | c_recover_ri(...)
        | c_recover_rc(...) -> select BS in (select BI in pdu)
  endcase
endfunc

function begin_trans_pdu_to_sp (cpdu: CPDU,ver: ccr_ver,
                aet1: ae_title,aet2: ae_title) : CSP is
        CBeginInd(get_aaimn(pdu,ver,aet1,aet2),
                            get_aais(pdu),
                            get_bisn(pdu,ver,aet1,aet2),
                            get_bis(pdu),
                            to_sp(get_ud(pdu)))
endfunc

function recover_trans_pdu_to_sp (cpdu: CPDU,ver: ccr_ver,
                aet1: ae_title,aet2: ae_title) : CSP is
  case pdu in
          c_recover_ri(...) -> CRecoverInd(get_rs(pdu), get_aaimn(pdu,ver,aet1,aet2),
                                get_aais(pdu), get_bisn(pdu,ver,aet1,aet2),
                                get_bis(pdu), to_sp(get_ud(pdu)))
        | c_recover_rc(...)  -> CRecoverCnf(get_rr(pdu), get_aaimn(pdu,ver,aet1,aet2),
                                get_aais(pdu), get_bisn(pdu,ver,aet1,aet2),
                                get_bis(pdu), to_sp(get_ud(pdu)))
  endcase
endfunc
endmod
======================================================================== *)
(*-------------------------------------------------------------*)
(*      Key Queue Type :                                       *)
(*      The element of a  sort key_queue is a list of keys.    *)
(*      The operators _._ and  _+_ on key_queue are used for   *)
(*      the operations of concatenation or deconcatenation of  *)
(*      PDUs in between CCR and the presentation layer.        *)
(*      In ConcDeconcType this operation plays an important    *)
(*      role when the operation "may_conc" is defined.         *)
(*-------------------------------------------------------------*)
type    Key_Queue is Key
sorts   key_queue
opns
        kq_empty (*! constructor *) : -> key_queue
        _._ (*! constructor *) : key,key_queue -> key_queue
        _+_ : key_queue,key -> key_queue
eqns
forall  k,k1:key,kq:key_queue
ofsort  key_queue
```

```
        kq_empty + k = k . kq_empty;
        (k . kq) + k1 = k . (kq + k1);
endtype (* Key_Queue *)
(* ----------------------------------------------------------------------------
module  Key_Queue is Key

type    key_queue is
          kq_empty
        | _._ (K: key,Kq: key_queue)
endtype

function _+_ (Kq: key_queue, K: key) : key_queue is
  case Kq in
          kq_empty -> K.kq_empty
        | (k: key).(kq: key_queue) -> k.(kq + K)
  endcase
endfunc
endmod
=============================================================================== *)
(*-------------------------------------------------------------*)
(*      PDU Type :                                             *)
(*      This is a general PDU type in this description.        *)
(*      The structure of a general PDU is similar to           *)
(*      a general service primitive. For the use of key_queue  *)
(*      in the definition of "may_conc" in ConcDeconcType,     *)
(*      the mapping function "h" into key_queue is defined.    *)
(*      The key is an abstraction of each general PDU.         *)
(*      The argument of general PDU, a real PDU, may have some *)
(*      parameters, but mapped result by "h", key, has no      *)
(*      parameter.                                             *)
(*-------------------------------------------------------------*)
type    Protocol_Data_Unit is CCR_PDU,A_PDU
sorts   PDU
opns
        empty (*! constructor *) : -> PDU
        pdu (*! constructor *) : CPDU -> PDU
        pdu (*! constructor *) : APDU -> PDU
        is_empty : PDU -> Bool
        is_ccr_pdu : PDU -> Bool
        is_ase_pdu : PDU -> Bool
        get_cpdu  : PDU -> CPDU
        get_apdu  : PDU -> APDU
        h : PDU -> key
eqns
forall  c:CPDU,a:APDU
ofsort  Bool
        is_empty(empty) = true;
        is_empty(pdu(c)) = false;
        is_empty(pdu(a)) = false;
        is_ccr_pdu(empty) = false;
        is_ccr_pdu(pdu(c)) = true;
        is_ccr_pdu(pdu(a)) = false;
        is_ase_pdu(empty) = false;
        is_ase_pdu(pdu(c)) = false;
        is_ase_pdu(pdu(a)) = true;
ofsort  CPDU
        get_cpdu(pdu(c)) = c;
ofsort  APDU
        get_apdu(pdu(a)) = a;
ofsort  key
        h(pdu(c)) = h(c);
        h(pdu(a)) = h(a);
endtype (* Protocol_Data_Unit *)
(* ------------------------------------------------------------------------
module  Protocol_Data_Unit is CCR_PDU, A_PDU
type    PDU is
          empty
        | pdu (C: CPDU)
        | pdu (A: APDU)
endtype

function is_empty (P: PDU) : Bool is P match empty endfunc

function is_ccr_pdu (P: PDU) : Bool is P match pdu (any CPDU) endfunc

function is_ase_pdu (P: PDU) : Bool is P match pdu (any APDU) endfunc

function get_cpdu  (P: PDU) : CPDU is select C in P endfunc

function get_apdu  (P: PDU) : APDU is  select A in P endfunc

/* no translation is needed for function h */
endmod
======================================================================== *)
(*-------------------------------------------------------------*)
(*      PDU Queue Type :                                       *)
(*      In this type, the quueue of PDUs and the operations for *)
(*      the queue are defined. The operation "h" is used for    *)
(*      the mapping of PDU queue to key quueue in the definition*)
(*      "may_conc" in ConcDeconcType.                          *)
(*-------------------------------------------------------------*)
type    PDU_Queue is Protocol_Data_Unit,Key_Queue
sorts   PDUqueue
opns
        q_empty (*! constructor *) : -> PDUqueue
        _._ (*! constructor *) : PDU,PDUqueue -> PDUqueue
        head : PDUqueue -> PDU
        tail : PDUqueue -> PDUqueue
        queue : PDU -> PDUqueue
        _+_ : PDUqueue,PDU -> PDUqueue
        is_q_empty,is_PDU : PDUqueue -> Bool
        h : PDUqueue -> key_queue
eqns
forall  p,p1:PDU,q:PDUqueue
ofsort  PDU
        head(p.q) = p;
ofsort  PDUqueue
        tail(p.q) = q;
        queue(empty) = q_empty;
        queue(p) = p.q_empty;
        q_empty + p = queue(p);
        (p.q) + p1 = p.(q + p1);
ofsort  bool
        is_q_empty(q_empty) = true;
        is_q_empty(p.q) = false;
        is_PDU(q) = not(is_q_empty(q)) and is_q_empty(tail(q));
ofsort  key_queue
        h(p.q) = h(p).h(q);
        h(q_empty) = kq_empty;
endtype (* PDU_Queue *)
(* ---------------------------------------------------------------------------
module  PDU_Queue is Protocol_Data_Unit, Key_Queue, User_Data
type    PDUqueue is
          q_empty
        | _._   (P: PDU,PQ: PDUqueue)
        | to_sp (UD: user_data)
endtype
function head (Q: PDUqueue) : PDU is select P in Q endfunc

function tail (Q: PDUqueue) : PDUqueue is select PQ in Q endfunc

function queue (P: PDU) : PDUqueue is
  case P in
          empty -> q_empty
        | any PDU -> P.q_empty
  endcase
endfunc

function _+_ (Q:PDUqueue, P:PDU) : PDUqueue is
  case Q is
          q_empty -> queue(P)
        | any PDUqueue -> (select P in Q).((select PQ in Q) + P)
  endcase
endfunc

function is_q_empty (Q: PDUqueue) : Bool is Q match q_empty endfunc

function is_PDU (Q: PDUqueue) : Bool is Q match (any PDU).q_empty endfunc

/* no translation is needed for function h */
endmod
=========================================================================== *)
(*-------------------------------------------------------------*)
(*      Search CCR PDU Type :                                  *)
(*      In the process part of this description,               *)
(*      the operation which searches a CCR PDU and which       *)
(*      obtains it is necessary. This type gives the operation. *)
(*-------------------------------------------------------------*)
type    Search_CCR_PDU is PDU_Queue
opns
        ccr_pdu : PDUqueue -> CPDU
        ase_pdu : PDUqueue -> PDUqueue
```

```
eqns
forall  p:PDU,pq:PDUqueue
ofsort  CPDU
        ccr_pdu(q_empty) = c_empty;
        is_ccr_pdu(p) =>
                ccr_pdu(p.pq) = get_cpdu(p);
        not(is_ccr_pdu(p)) =>
                ccr_pdu(p.pq) = ccr_pdu(pq);
ofsort  PDUqueue
        ase_pdu(q_empty) = q_empty;
        is_ccr_pdu(p) =>        ase_pdu(p.pq) = ase_pdu(pq);
        not(is_ccr_pdu(p)) =>   ase_pdu(p.pq) = p.ase_pdu(pq);
endtype (* Search_CCR_PDU *)
(* ------------------------------------------------------------------------
module  Search_CCR_PDU is PDU_Queue

function ccr_pdu (PQ: PDUqueue) : CPDU is
  case PQ in
          q_empty -> c_empty
        | any PDUqueue -> let p:PDU=select P in PQ, q:PDUqueue=select PQ in PQ in
                                case p in
                                          pdu (C:=C :CPDU) -> C
                                        | any PDU -> ccr_pdu(q)
                                endcase
  endcase
endfunc

function ase_pdu (PQ: PDUqueue) : PDUqueue is
  case PQ in
          q_empty -> q_empty
        | any PDUqueue -> let p:PDU=select P in PQ, q:PDUqueue=select PQ in PQ in
                                case p in
                                          pdu (A:=A :APDU) -> A
                                        | any PDU -> p.ase_pdu(q)
                                endcase
  endcase
endfunc
endmod
======================================================================== *)
(*-------------------------------------------------------------*)
(*      Concatination and Deconcatination Type :               *)
(*      The operator may_conc is a predicate: it is true       *)
(*      if the PDUs may be concatenated. This rule should be   *)
(*      the same as the description of ISO/IEC IS 9805         *)
(*      chapter 10.                                            *)
(*-------------------------------------------------------------*)
type    ConcDeconcType is Protocol_Data_Unit,PDU_Queue
opns
        may_conc : key_queue,key -> Bool
        is_empty_seq, is_apdu_seq,
        is_c_begin_ri_seq, is_c_begin_rc_seq,
        is_c_prepare_ri_seq, is_c_ready_ri_seq,
        is_c_commit_ri_seq, is_c_commit_rc_seq, is_c_commit_rc_seq_tail,
        is_c_rollback_ri_seq, is_c_rollback_rc_seq,
        is_c_begin_ri_pdu, is_c_begin_rc_pdu : key_queue -> Bool
eqns
forall  k:key,kq:key_queue
ofsort  Bool
        may_conc(kq,k)  =  is_empty_seq(kq)
                        or is_apdu_seq(kq + k)
                        or is_c_begin_ri_seq(kq + k)
                        or is_c_begin_rc_seq(kq + k)
                        or is_c_prepare_ri_seq(kq + k)
                        or is_c_ready_ri_seq(kq + k)
                        or is_c_commit_ri_seq(kq + k)
                        or is_c_commit_rc_seq(kq + k)
                        or is_c_rollback_ri_seq(kq + k)
                        or is_c_rollback_rc_seq(kq + k);

        is_empty_seq(kq_empty)  = true;
        is_empty_seq(k.kq)      = false;

        is_apdu_seq(kq_empty)   = true;
        is_apdu_seq(k.kq)       = (k eq apdu) and is_apdu_seq(kq);

        is_c_begin_ri_seq(kq_empty)     = false;
        is_c_begin_ri_seq(k.kq)         =
                ((k eq apdu) and is_c_begin_ri_seq(kq)) or
                ((k eq c_begin_ri) and
                        (is_c_prepare_ri_seq(kq) or is_apdu_seq(kq)));

        is_c_begin_rc_seq(kq_empty)     = false;
        is_c_begin_rc_seq(k.kq)         =
                ((k eq apdu) and is_c_begin_rc_seq(kq)) or
                ((k eq c_begin_rc) and
                        (is_c_ready_ri_seq(kq) or
                                (is_c_ready_ri_seq(kq) or is_apdu_seq(kq))));

        is_c_prepare_ri_seq(kq_empty)   = false;
        is_c_prepare_ri_seq(k.kq)       =
                ((k eq apdu) and is_c_prepare_ri_seq(kq)) or
                ((k eq c_prepare_ri) and is_empty_seq(kq));

        is_c_ready_ri_seq(kq_empty)     = false;
        is_c_ready_ri_seq(k.kq)         =
                ((k eq apdu) and is_c_ready_ri_seq(kq)) or
                ((k eq c_ready_ri) and is_empty_seq(kq));

        is_c_commit_ri_seq(kq_empty)    = false;
        is_c_commit_ri_seq(k.kq)        =
                (k eq c_commit_ri) and is_c_begin_ri_pdu(kq);

        is_c_commit_rc_seq(k.kq)        =
                (k eq c_commit_rc) and is_c_commit_rc_seq_tail(kq);

        is_c_commit_rc_seq_tail(kq_empty)       = false;
        is_c_commit_rc_seq_tail(k.kq)           =
                ((k eq c_begin_rc) and
                        (is_c_ready_ri_seq(kq) or is_apdu_seq(kq))) or
                is_c_ready_ri_seq(k.kq) or is_apdu_seq(k.kq);

        is_c_rollback_ri_seq(kq_empty)  = false;
        is_c_rollback_ri_seq(k.kq)      =
                (k eq c_rollback_ri) and is_c_begin_ri_pdu(kq);

        is_c_rollback_rc_seq(kq_empty)  = false;
        is_c_rollback_rc_seq(k.kq)      =
                (k eq c_rollback_rc) and
                        (is_c_begin_ri_pdu(kq) or is_c_begin_rc_pdu(kq) or
                         is_apdu_seq(kq));

        is_c_begin_ri_pdu(kq_empty)     = false;
        is_c_begin_ri_pdu(k.kq)         =
                (k eq c_begin_ri) and is_empty_seq(kq);

        is_c_begin_rc_pdu(kq_empty)     = false;
        is_c_begin_rc_pdu(k.kq)         =
                (k eq c_begin_rc) and is_empty_seq(kq);
endtype (* ConcDeconcType *)
(* ------------------------------------------------------------------------
module  ConcDeconcType is Protocol_Data_Unit, PDU_Queue
/* to avoid h use Protocol_Data_Unit, PDU_Queue */

/* function may_conc (KQ: key_queue, K: key) : Bool is
  is_empty_seq(KQ)
  or is_apdu_seq(KQ + K)
  or is_c_begin_ri_seq(KQ + K)
  or is_c_begin_rc_seq(KQ + K)
  or is_c_prepare_ri_seq(KQ + K)
  or is_c_ready_ri_seq(KQ + K)
  or is_c_commit_ri_seq(KQ + K)
  or is_c_commit_rc_seq(KQ + K)
  or is_c_rollback_ri_seq(KQ + K)
  or is_c_rollback_rc_seq(KQ + K)
endfunc
*/
 function may_conc (PQ: PDUqueue,P: PDU) : Bool is
        is_q_empty(PQ)
        or is_apdu_seq(PQ+P)
        or is_c_begin_ri_seq(PQ+P)
        or is_c_begin_rc_seq(PQ+P)
        or is_c_prepare_ri_seq(PQ+P)
        or is_c_ready_ri_seq(PQ + P)
        or is_c_commit_ri_seq(PQ + P)
        or is_c_commit_rc_seq(PQ + P)
        or is_c_rollback_ri_seq(PQ + P)
        or is_c_rollback_rc_seq(PQ + P)
endfunc
/*
function is_empty_seq (KQ: key_queue) : Bool is KQ match kq_empty endfunc

function is_apdu_seq (KQ: key_queue) : Bool is
```

```
  case  KQ in
          kq_empty -> true
        | (k:key).(q:key_queue) -> (k eq apdu) and is_apdu_seq(q)
  endcase
endfunc
*/
function is_apdu_seq (Q: PDUqueue) : Bool is
  case  Q in
          q_empty -> true
        | any PDUqueue -> is_ase_pdu(select P in Q) and is_apdu_seq(select PQ in Q)
  endcase
endfunc
/*
function is_c_begin_ri_seq(KQ: key_queue) : Bool is
  case  KQ in
          kq_empty -> false
        | (k:key).(q:key_queue) ->
                ((k eq apdu) and is_c_begin_ri_seq(q)) or
                ((k eq c_begin_ri) and
                        (is_c_prepare_ri_seq(q) or is_apdu_seq(q)))
  endcase
endfunc
*/
function is_c_begin_ri_seq(Q: PDUqueue) : Bool is
  case  Q in
          q_empty -> false
        | any PDUqueue -> let p:PDU=select P in Q, q:PDUqueue=select PQ in Q in
                (is_ase_pdu(p) and is_c_begin_ri_seq(q)) or
                ((is_ccr_pdu(p) and is_c_begin_ri(p)) and
                        (is_c_prepare_ri_seq(q) or is_apdu_seq(q)))
  endcase
endfunc
*/
function is_c_prepare_ri_seq (KQ: key_queue) : Bool is
  case  KQ in
          kq_empty -> false
        | (k:key).(q:key_queue) ->
                ((k eq apdu) and is_c_prepare_ri_seq(q)) or
                ((k eq c_prepare_ri) and is_empty_seq(q))
  endcase
endfunc
*/
function is_c_prepare_ri_seq (Q: PDUqueue) : Bool is
  case  Q in
          q_empty -> false
        | any PDUqueue -> let p:PDU=select P in Q, q:PDUqueue=select PQ in Q in
                (is_ase_pdu(p) and is_c_prepare_ri_seq(q)) or
                ((is_ccr_pdu(p) and is_c_prepare_ri(p)) and is_q_empty(q))
  endcase
endfunc
/*
function is_c_ready_ri_seq (KQ: key_queue) : Bool is
  case  KQ in
          kq_empty -> false
        | (k:key).(q:key_queue) ->
                ((k eq apdu) and is_c_ready_ri_seq(q)) or
                ((k eq c_ready_ri) and is_empty_seq(q))
  endcase
endfunc
*/
function is_c_ready_ri_seq (Q: PDUqueue) : Bool is
  case  Q in
          q_empty -> false
        | any PDUqueue -> let p:PDU=select P in Q, q:PDUqueue=select PQ in Q in
                (is_ase_pdu(p) and is_c_ready_ri_seq(q)) or
                ((is_ccr_pdu(p) and is_c_ready_ri(p)) and is_q_empty(q))
  endcase
endfunc
/*
function is_c_commit_ri_seq (KQ: key_queue) : Bool is
  case  KQ in
          kq_empty -> false
        | (k:key).(q:key_queue) ->
                (k eq c_commit_ri) and is_c_begin_ri_pdu(q)
  endcase
endfunc
*/
function is_c_commit_ri_seq (Q: PDUqueue) : Bool is
  case  Q in
          q_empty -> false
        | any PDUqueue -> let p:PDU=select P in Q, q:PDUqueue=select PQ in Q in
                ((is_ccr_pdu(p) and is_c_commit_ri(p)) and is_c_begin_ri_pdu(q))
  endcase
endfunc
/*
function is_c_commit_rc_seq (KQ: key_queue) : Bool is
  case  KQ in
          kq_empty -> false
         (k:key).(q:key_queue) ->
                (k eq c_commit_rc) and is_c_commit_rc_seq_tail(q)
  endcase
endfunc
*/
function is_c_commit_rc_seq (Q: PDUqueue) : Bool is
  case  Q in
          q_empty -> false
        | any PDUqueue -> let p:PDU=select P in Q, q:PDUqueue=select PQ in Q in
                ((is_ccr_pdu(p) and is_c_commit_rc(p)) and is_c_commit_rc_seq_tail(q))
  endcase
endfunc
/*
function is_c_commit_rc_seq_tail (KQ: key_queue) : Bool is
  case  KQ in
          kq_empty -> false
        | (k:key).(q:key_queue) ->
                ((k eq c_begin_rc) and
                 (is_c_ready_ri_seq(q) or is_apdu_seq(q))) or
                  is_c_ready_ri_seq(KQ) or is_apdu_seq(KQ)
  endcase
endfunc
*/
function is_c_commit_rc_seq_tail (Q: PDUqueue) : Bool is
  case  Q in
          q_empty -> false
        | any PDUqueue -> let p:PDU=select P in Q, q:PDUqueue=select PQ in Q in
             ((is_ccr_pdu(p) and is_c_begin_rc(p)) and
                (is_c_ready_ri_seq(q) or is_apdu_seq(q))) or
                 is_c_ready_ri_seq(KQ) or is_apdu_seq(KQ)
  endcase
endfunc
/*
function is_c_rollback_ri_seq (KQ: key_queue) : Bool is
  case  KQ in
          kq_empty -> false
        | (k:key).(q:key_queue) ->
                (k eq c_rollback_ri) and is_c_begin_ri_pdu(q)
  endcase
endfunc
*/
function is_c_rollback_ri_seq (Q: PDUqueue) : Bool is
  case  Q in
          q_empty -> false
        | any PDUqueue -> let p:PDU=select P in Q, q:PDUqueue=select PQ in Q in
             ((is_ccr_pdu(p) and is_c_rollback_ri(p)) and is_c_begin_ri_pdu(q)
  endcase
endfunc
/*
function is_c_rollback_rc_seq (KQ: key_queue) : Bool is
  case  KQ in
          kq_empty -> false
        | (k:key).(q:key_queue) ->
                (k eq c_rollback_rc) and
                (is_c_begin_ri_pdu(q) or is_c_begin_rc_pdu(q) or
                 is_apdu_seq(q))
  endcase
endfunc
*/
function is_c_rollback_rc_seq (Q: PDUqueue) : Bool is
  case  Q in
          q_empty -> false
        | any PDUqueue -> let p:PDU=select P in Q, q:PDUqueue=select PQ in Q in
             ((is_ccr_pdu(p) and is_c_rollback_rc(p)) and
                        (is_c_begin_ri_pdu(q) is_c_begin_rc_pdu(q) or is_apdu_seq(q))
  endcase
endfunc
/*
function is_c_begin_ri_pdu (KQ: key_queue) : Bool is
  case  KQ in
          kq_empty -> false
        | (k:key).(q:key_queue) ->
                (k eq c_begin_ri) and is_empty_seq(kq)
  endcase
```

```
endfunc
*/
function is_c_begin_ri_pdu (Q: PDUqueue) : Bool is
  case  Q in
          q_empty -> false
        | any PDUqueue -> let p:PDU=select P in Q, q:PDUqueue=select PQ in Q in
            ((is_ccr_pdu(p) and is_c_begin_ri(p)) and is_empty_seq(q)
  endcase
endfunc
/*
function is_c_begin_rc_pdu (KQ: key_queue) : Bool is
  case  KQ in
          kq_empty -> false
        | (k:key).(q:key_queue) ->
                (k eq c_begin_rc) and is_empty_seq(kq)
  endcase
endfunc
*/
function is_c_begin_rc_pdu (Q: PDUqueue) : Bool is
  case  Q in
          q_empty -> false
        | any PDUqueue -> let p:PDU=select P in Q, q:PDUqueue=select PQ in Q in
            ((is_ccr_pdu(p) and is_c_begin_rc(p)) and is_empty_seq(q)
  endcase
endfunc
endmod
====================================================================== *)
(*------------------------------------------------------------*)
(*      CCR Version Number Types :                            *)
(*      This type includes the definition of CCR version       *)
(*      numbers. Four elements are defined as follows;         *)
(*      ver1 ----- the protocol machine supports only version 1 *)
(*      ver2 ----- the protocol machine supports only version 2 *)
(*      ver12 ---- the protocol machine supports               *)
(*                              both version 1 and version 2    *)
(*      rejected - it represents the failure of version        *)
(*                      negatiation between peer AEIs.         *)
(*      The operator "select_ver" returns the negotiated       *)
(*      version number.                                        *)
(*------------------------------------------------------------*)
type    CCR_Version_Number is Boolean,Key
sorts   ccr_ver
opns
        ver1 (*! constructor *),
        ver2 (*! constructor *),
        ver12 (*! constructor *),
        rejected (*! constructor *) : -> ccr_ver
        select_ver : ccr_ver,ccr_ver -> ccr_ver
        _contains_ : ccr_ver,ccr_ver -> bool
        _eq_,_ne_ : ccr_ver,ccr_ver -> bool
        h : ccr_ver -> key
eqns
forall  ver,vers:ccr_ver
ofsort  key
```

```
        h(rejected) = 0;
        h(ver1) = succ(h(rejected));
        h(ver2) = succ(h(ver1));
        h(ver12) = succ(h(ver2));
ofsort  Bool
        ver eq vers = h(ver) eq h(vers);
        ver ne vers = not(ver eq vers);
        ver12 contains ver = ver ne rejected;
        vers ne ver12 =>
                vers contains ver = vers eq ver;
ofsort  ccr_ver
        (ver contains ver2) and (vers contains ver2) =>
                select_ver(ver,vers) = ver2;
        not((ver contains ver2) and (vers contains ver2))
        and (ver contains ver1) and (vers contains ver1) =>
                select_ver(ver,vers) = ver1;
        not((ver contains ver2) and (vers contains ver2))
        and not((ver contains ver1) and (vers contains ver1)) =>
                select_ver(ver,vers) = rejected;
endtype (* CCR_Version_Number *)
(* ---------------------------------------------------------------------------
module  CCR_Version_Number is Boolean, Key

type    ccr_ver is
         ver1
        | ver2
        | ver12
        | rejected
endtype

function select_ver (VER1: ccr_ver, VER2: ccr_ver) : ccr_ver is
  if (ver contains ver2) and (vers contains ver2) then ver2
  elsif not((ver contains ver2) and (vers contains ver2))
        and (ver contains ver1) and (vers contains ver1) then  ver1
  elsif not((ver contains ver2) and (vers contains ver2))
        and not((ver contains ver1) and (vers contains ver1)) then rejected
  endcase
endfunc

function _contains_ (VER1: ccr_ver, VER2: ccr_ver) : bool is
  case VER1 in
          ver12 -> VER2 ne rejected
        | vers: ccr_ver when vers ne ver12 -> vers eq VER2
  endcase
endfunc

/* no translation is needed for function h */
endmod
========================================================================= *)
(*-------------------------------------------------------------*)
(*      Atomic Action Branch Type :                            *)
(*      This type represents Atomic Action Branch.             *)
(*      Atomic Action Branch may be omitted, so "aab_null" is   *)
(*      also defined.                                          *)
```

```
(*------------------------------------------------------------*)
type    Atomic_Action_Branch is Atomic_Action_Identifier,Branch_Identifier,
                                       CCR_PDU_To_CCR_Service_Primitive
sorts   atomic_action_branch
opns
        aab_null (*! constructor *) : -> atomic_action_branch
        atomic_action_branch (*! constructor *) : atomic_action_identifier,
                              branch_identifier -> atomic_action_branch
        get_aai : atomic_action_branch -> atomic_action_identifier
        get_bi : atomic_action_branch -> branch_identifier
        _eq_,_ne_ : atomic_action_branch,atomic_action_branch -> Bool
        get_aab :  CPDU,ccr_ver,ae_title,ae_title -> atomic_action_branch
        get_aab :  CSP  -> atomic_action_branch
eqns
forall  aab,aab1:atomic_action_branch,
        pdu:CPDU,ver:ccr_ver,aet1,aet2:ae_title,csp:CSP,
        aai,aai1:atomic_action_identifier,bi,bi1:branch_identifier
ofsort  atomic_action_identifier
        get_aai(atomic_action_branch(aai,bi)) = aai;
ofsort  branch_identifier
        get_bi(atomic_action_branch(aai,bi)) = bi;
ofsort  Bool
        aab_null eq aab_null = true;
        aab_null eq atomic_action_branch(aai,bi) = false;
        atomic_action_branch(aai,bi) eq aab_null = false;
        atomic_action_branch(aai,bi) eq atomic_action_branch(aai1,bi1)
                = (aai eq aai1) and (bi eq bi1);
        aab ne aab1 = not(aab eq aab1);
ofsort  atomic_action_branch
        get_aab(pdu,ver,aet1,aet2) =
                atomic_action_branch(
                        atomic_action_identifier(
                                masters_name(get_aaimn(pdu,ver,aet1,aet2)),
                                get_aais(pdu)),
                        branch_identifier(
                                superiors_name(get_bisn(pdu,ver,aet1,aet2)),
                                get_bis(pdu)));
        get_aab(csp) =
                atomic_action_branch(
                        atomic_action_identifier(
                                masters_name(get_aaimn(csp)),
                                get_aais(csp)),
                        branch_identifier(
                                superiors_name(get_bisn(csp)),
                                get_bis(csp)));
endtype (* Atomic_Action_Branch *)
(* ---------------------------------------------------------------------
module  Atomic_Action_Branch is Atomic_Action_Identifier,
                                Branch_Identifier,
                                CCR_PDU_To_CCR_Service_Primitive
type    atomic_action_branch is
          aab_null
        | atomic_action_branch (AAI: atomic_action_identifier,
                                BI: branch_identifier)
endtype

function get_aai (AB: atomic_action_branch) : atomic_action_identifier is
  select AAI in AB
endfunc

function get_bi (AB: atomic_action_branch) : atomic_action_identifier is
  select BI in AB
endfunc

function get_aab (pdu: CPDU,ver: ccr_ver,aet1: ae_title,
                aet2: ae_title) : atomic_action_branch is
  atomic_action_branch(
        atomic_action_identifier(
                masters_name(get_aaimn(pdu,ver,aet1,aet2)),
                get_aais(pdu)),
        branch_identifier(
                superiors_name(get_bisn(pdu,ver,aet1,aet2)),
                get_bis(pdu)))
endfunc

function get_aab (csp: CSP) : atomic_action_branch is
  atomic_action_branch(
        atomic_action_identifier(
                masters_name(get_aaimn(csp)),
                get_aais(csp)),
        branch_identifier(
                superiors_name(get_bisn(csp)),
                get_bis(csp)))
endfunc
endmod
========================================================================== *)
(*---------------------------------------------------------------*)
(*      Failure Event Type :                                     *)
(*      "Failure" represents a failure event. This failure is    *)
(*      general. In CCR protocol specification, the kind of      *)
(*      failures is not significant.                             *)
(*---------------------------------------------------------------*)
type    Failure is
sorts   failure
opns
        failure (*! constructor *) : -> failure
endtype (* Failure *)
(* ------------------------------------------------------------------------
module  Failure is
type    failure is
        failure
endtype
endmod
========================================================================== *)
(*---------------------------------------------------------------*)
(*      End of Type Definitions                                  *)
(*---------------------------------------------------------------*)
```

```
(*-------------------------------------------------------------*)
(*      Behaviour Definitions                                  *)
(*-------------------------------------------------------------*)
(*-------------------------------------------------------------*)
(*      CCR Protocol Specification Behaviour :                 *)
(*      Two gates ccr_u and ccr_l are used to communicate in   *)
(*      between two processes CF (Control Function) and CCR    *)
(*      (CCR protocol machine).                                *)
(*-------------------------------------------------------------*)
behaviour
        hide ccr_u,ccr_l in
        ( CCR[ccr_u,ccr_l,env](ver)
          |[ccr_u,ccr_l,env]|
          CF[u,p,ccr_u,ccr_l,acse_u,acse_l,env,u_env]
        )
where

(*-------------------------------------------------------------*)
(*      CCR Protocol Machine :                                 *)
(*      Two cases are described in process CCR. The first is   *)
(*      the case that the negotiation of CCR protocol version  *)
(*      is necessary. The process CCRVersionNegotiate works for *)
(*      this purpose. The second case is that the negotiation  *)
(*      of version numbers finished in advance and that        *)
(*      the environment (e.g. CCR user) knows the negotiated   *)
(*      version. This case may occur after some failures       *)
(*      happened. It is treared by the process CCRFixedVersion. *)
(*-------------------------------------------------------------*)
process CCR[c_u,c_l,env](ver:ccr_ver) : exit :=
        (         CCRVersionNegotiate[c_u,c_l](ver)
          []
          ( env?fixed_ver:ccr_ver?aet_self:AE_title?aet_peer:AE_title;
            CCRFixedVersion[c_u,c_l](fixed_ver,aet_self,aet_peer)
        ) ) [> env!failure;exit
where

(*-------------------------------------------------------------*)
(*      CCR Protocol Version Negotiation                       *)
(*      The negotiation of CCR protocol version is necessary   *)
(*      when the protocol machine has functionalities of       *)
(*      version 2. If it has only functionalities of version 1, *)
(*      the negotiation is meaingless. In the process          *)
(*      CCRInitialize the negotiation of version is carried out.*)
(*      After that CCRPM can start.                            *)
(*-------------------------------------------------------------*)
process CCRVersionNegotiate[c_u,c_l](ver:ccr_ver) : exit :=
        c_u!ver;
        ( ( [ver eq ver1]->exit(ver1)
            []
            [ver contains ver2]->CCRInitialize[c_u,c_l](ver)
          ) >> accept fixed_ver:ccr_ver in
                ( [fixed_ver eq rejected]->exit
                  []
                  [fixed_ver ne rejected]->
```

```
                                c_u?aet_self:AE_title?aet_peer:AE_title;
                                CCRPM[c_u,c_l](fixed_ver,aet_self,aet_peer)
        )       )
where

(*-------------------------------------------------------------*)
(*      Process for CCR Initialize                             *)
(*      CCRIitialize negotiates CCR protocol version. Two PDUs, *)
(*      c_initiaize_ri and c_initialize_rc, concern the        *)
(*      negotiation. If the peer entity has only version 1,     *)
(*      these PDUs are not accepted and the response PDU is     *)
(*      empty.                                                 *)
(*-------------------------------------------------------------*)
process CCRInitialize[c_u,c_l](ver:ccr_ver) : exit(ccr_ver) :=
        ( c_u!AAssoc_req;
          c_l!c_initialize_ri(ver);
          ( ( c_l?pdu:CPDU[is_c_initialize_rc(pdu)];
              c_u!select_ver(ver,get_v(pdu));
              exit(select_ver(ver,get_v(pdu)))
            )
            []
            ( c_l!c_empty;
              c_u!select_ver(ver,ver1);
              exit(select_ver(ver,ver1))
        ) ) )
        []
        ( ( ( c_l?pdu:CPDU[is_c_initialize_ri(pdu)];
              c_u!select_ver(ver,get_v(pdu));
              stop
             )
            []
            ( c_l!c_empty;
              c_u!select_ver(ver,ver1);
              stop
          ) )
          [> ( c_u?fixed_ver:ccr_ver;
               ( [fixed_ver eq rejected]->exit(fixed_ver)
                 []
                 [fixed_ver ne rejected]->
                 ( c_u!AAssoc_rsp;
                   c_l!c_initialize_rc(fixed_ver);
                   exit(fixed_ver)
        ) ) ) )
endproc (* CCRInitialize *)

endproc (* CCRVersionNegotiate *)

(*-------------------------------------------------------------*)
(*      CCR Fixed Version Process :                            *)
(*      When the negotiation of CCR protocol version finished   *)
(*      in advance and the environment (e.g. CCR user) knows    *)
(*      an available version, this process works. This case     *)
(*      happens, e.g., in the recovery case.                   *)
(*-------------------------------------------------------------*)
```

```
process CCRFixedVersion[c_u,c_l](ver:ccr_ver,aet_self,aet_peer:ae_title)
                                                               : noexit :=
        CCRPM[c_u,c_l](ver,aet_self,aet_peer)
endproc (* CCRFixedVersion *)

(*-------------------------------------------------------------*)
(*      Process for CCR Behaviour :                            *)
(*      This process describes the behaviour of a CCR protocol  *)
(*      machine.                                               *)
(*-------------------------------------------------------------*)
process CCRPM[c_u,c_l](ver:ccr_ver,aet_self,aet_peer:ae_title) : noexit :=
        (    SupCCR[c_u,c_l](ver,aet_self,aet_peer)
          [] SubCCR[c_u,c_l](ver,aet_self,aet_peer)
        ) >> CCRPM[c_u,c_l](ver,aet_self,aet_peer)
where

(*-------------------------------------------------------------*)
(*      Process for Superior CCR End Point :                   *)
(*-------------------------------------------------------------*)
process SupCCR[c_u,c_l](ver:ccr_ver,aet_self,aet_peer:ae_title) : exit :=
        SupAction[c_u,c_l](ver,aet_self,aet_peer,aab_null,aab_null)
        []
        SupRecovery[c_u,c_l](ver,aet_self,aet_peer,aab_null,aab_null)
where

(*-------------------------------------------------------------*)
(*      Process for Superior CCRAction : Normal                *)
(*      C-BEGIN req                                            *)
(*-------------------------------------------------------------*)
process SupAction[c_u,c_l](ver:ccr_ver,aet_self,aet_peer:ae_title,
                                c_br,n_br:atomic_action_branch) : exit :=
        c_u?sp:CSP[is_CBeginReq(sp)];
        c_l!begin_trans_sp_to_pdu(sp,ver,aet_self,aet_peer);
        ContSupAction[c_u,c_l](ver,aet_self,aet_peer,get_aab(sp),n_br)
where

(*-------------------------------------------------------------*)
(*      Process for Superior CCR Action : Normal               *)
(*      C-BEGIN cnf ..... C-PREPARE req                        *)
(*-------------------------------------------------------------*)
process ContSupAction[c_u,c_l](ver:ccr_ver,aet_self,aet_peer:ae_title,
                                c_br,n_br:atomic_action_branch): exit :=
        ( c_l?pdu:CPDU[is_c_begin_rc(pdu)];
          c_u!CBeginCnf(to_sp(get_ud(pdu)));
          ( ( c_u?sp:CSP[is_CPrepareReq(sp)];
              c_l!c_prepare_ri(to_pdu(get_ud(sp)));
              ContSupAction1[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
            )
            []
            ContSupAction1[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
        ) )
        []
        ( c_u?sp:CSP[is_CPrepareReq(sp)];
          c_l!c_prepare_ri(to_pdu(get_ud(sp)));
          ( ( c_l?pdu:CPDU[is_c_begin_rc(pdu)];
              c_u!CBeginCnf(to_sp(get_ud(pdu)));
              ContSupAction1[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
            )
            []
            ContSupAction1[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
        ) )
        []
        ContSupAction1[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
where

(*------------------------------------------------------------*)
(*      Process for Superior CCR Action : Normal              *)
(*      C-READY req or C-ROLLBACK                             *)
(*------------------------------------------------------------*)
process ContSupAction1[c_u,c_l](ver:ccr_ver,aet_self,aet_peer:ae_title,
                                c_br,n_br:atomic_action_branch): exit :=
        ( c_l?pdu:CPDU[is_c_ready_ri(pdu)];
          c_u!CReadyInd(to_sp(get_ud(pdu)));
          SupDecision[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
        )
        []
        ( c_l?pdu:CPDU[is_c_rollback_ri(pdu)];
          c_u!CRollbackInd(to_sp(get_ud(pdu)));
          c_u?sp:CSP[is_CRollbackRsp(sp)];
          c_l!c_rollback_rc(to_pdu(get_ud(sp)));
          exit
        )
        []
        ( c_u?sp:CSP[is_CRollbackReq(sp)];
          c_l!c_rollback_ri(to_pdu(get_ud(sp)));
          SupRollSend1[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
        )
        []
        (*      When C-Rollback request + C-Begin request is recived,   *)
        (*      CCRPM simultaneously issues c-rollback-ri and           *)
        (*      c-begin-ri at c_l gate.                                 *)
        ( c_u?sp:CSP[is_CRollbackReq_CBeginReq(sp)];
          c_l!c_rollback_ri(to_pdu(get_ud(sp)))
             !begin_trans_sp_to_pdu(sp,ver,aet_self,aet_peer);
          SupRollsend2[c_u,c_l](ver,aet_self,aet_peer,c_br,get_aab(sp))
        )
where

(*------------------------------------------------------------*)
(*      Process for Superior CCR Action : Normal              *)
(*      C-COMMIT req or C-ROLLBACK req                        *)
(*------------------------------------------------------------*)
process SupDecision[c_u,c_l](ver:ccr_ver,aet_self,aet_peer:ae_title,
                                c_br,n_br:atomic_action_branch) : exit :=
        ( c_u?sp:CSP[is_CCommitReq(sp)];
          c_l!c_commit_ri(to_pdu(get_ud(sp)));
          c_l?pdu:CPDU[is_c_commit_rc(pdu)];
          c_u!CCommitCnf(to_sp(get_ud(pdu)));
          exit
        )
        []
        ( c_u?sp:CSP[is_CRollbackReq(sp)];
          c_l!c_rollback_ri(to_pdu(get_ud(sp)));
          c_l?pdu:CPDU[is_c_rollback_rc(pdu)];
          c_u!CRollbackCnf(to_sp(get_ud(pdu)));
          exit
        )
        []
        (*      When C-Commit request + C-Begin request is recived,     *)
        (*      CCRPM simultaneously issues c-commit-ri and c-begin-ri  *)
        (*      at c_l gate.                                            *)
        ( c_u?sp:CSP[is_CCommitReq_CBeginReq(sp)];
          c_l!c_commit_ri(to_pdu(get_ud(sp)))
             !begin_trans_sp_to_pdu(sp,ver,aet_self,aet_peer);
          c_l?pdu:CPDU[is_c_commit_rc(pdu)];
          c_u!CCommitCnf(to_sp(get_ud(pdu)));
          ContSupAction[c_u,c_l](ver,aet_self,aet_peer,get_aab(sp),aab_null)
        )
        []
        (*      When C-Rollback request + C-Begin request is recived,   *)
        (*      CCRPM simultaneously issues c-rollback-ri and           *)
        (*      c-begin-ri at c_l gate.                                 *)
        ( c_u?sp:CSP[is_CRollbackReq_CBeginReq(sp)];
          c_l!c_rollback_ri(to_pdu(get_ud(sp)))
             !begin_trans_sp_to_pdu(sp,ver,aet_self,aet_peer);
          c_l?pdu:CPDU[is_c_rollback_rc(pdu)];
          c_u!CRollbackCnf(to_sp(get_ud(pdu)));
          ContSupAction[c_u,c_l](ver,aet_self,aet_peer,get_aab(sp),aab_null)
        )
endproc (* SupDecision *)

(*------------------------------------------------------------*)
(*      Process for Superior CCR Action : Normal              *)
(*      C-ROLLBACK-RC or C-ROLLBACK collision                 *)
(*                              After C-ROLLBACK req          *)
(*------------------------------------------------------------*)
process SupRollSend1[c_u,c_l](ver:ccr_ver,aet_self,aet_peer:ae_title,
                              c_br,n_br:Atomic_Action_Branch) : exit :=
        [ver eq ver2]->
        ( c_l?pdu:CPDU[is_c_begin_rc(pdu)];
          SupRollSend1[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
        )
        []
        ( c_l?pdu:CPDU[is_c_rollback_rc(pdu)];
          c_u!CRollbackCnf(to_sp(get_ud(pdu)));
          exit
        )
        []
        ( c_l?pdu:CPDU[is_c_rollback_ri(pdu)];
          c_u!CRollbackInd(to_sp(get_ud(pdu)));
          c_u?sp:CSP[is_CRollbackRsp(sp)];
          c_l!c_rollback_rc(to_pdu(get_ud(sp)));
          exit
        )
endproc (* SupRollSend1 *)

(*-------------------------------------------------------------*)
(*      Process for Superior CCR Action : Normal               *)
(*      C-ROLLBACK-RC or C-ROLLBACK collision                  *)
(*                      After C-ROLLBACK req + C-BEGIN req      *)
(*-------------------------------------------------------------*)
process SupRollSend2[c_u,c_l](ver:ccr_ver,aet_self,aet_peer:ae_title,
                                c_br,n_br:Atomic_Action_Branch) : exit :=
        ( c_l?pdu:CPDU[is_c_rollback_rc(pdu)];
          c_u!CRollbackCnf(to_sp(get_ud(pdu)));
          ContSupAction[c_u,c_l](ver,aet_self,aet_peer,n_br,aab_null)
        )
        []
        ( c_l?pdu:CPDU[is_c_rollback_ri(pdu)];
          c_u!CRollbackInd(to_sp(get_ud(pdu)));
          c_u?sp:CSP[is_CRollbackRsp(sp)];
          c_l!c_rollback_rc(to_pdu(get_ud(sp)));
          c_l!begin_trans_sp_to_pdu(sp,ver,aet_self,aet_peer);
          ContSupAction1[c_u,c_l](ver,aet_self,aet_peer,n_br,aab_null)
        )
endproc (* SupRollSend2 *)

endproc (* ContSupAction1 *)

endproc (* ContSupAction *)

endproc (* SupAction *)

(*-------------------------------------------------------------*)
(*      Process for Superior CCR Recovery : Recover            *)
(*      C-RECOVER req or C-RECOVER-RI                          *)
(*-------------------------------------------------------------*)
process SupRecovery[c_u,c_l](ver:ccr_ver,aet_self,aet_peer:AE_title,
                                c_br,n_br:Atomic_Action_Branch) : exit :=
        ( c_l?pdu:CPDU[is_c_recover_ri(pdu) and (get_rs(pdu) eq ready)];
          c_u!recover_trans_pdu_to_sp(pdu,ver,aet_self,aet_peer);
          SupRecResponse[c_u,c_l](ver,aet_self,aet_peer,
                        get_aab(pdu,ver,aet_self,aet_peer),aab_null)
        )
        []
        ( c_u?sp:CSP[is_CRecoverReq(sp) and (get_rs(sp) eq commit)];
          c_l!recover_trans_sp_to_pdu(sp,ver,aet_self,aet_peer);
          SupRecActions[c_u,c_l](ver,aet_self,aet_peer,get_aab(sp),aab_null)
        )
where

(*-------------------------------------------------------------*)
(*      Process for Superior CCR Recovery : Recover            *)
(*      C-RECOVER(ready)-RI APDU received                      *)
(*-------------------------------------------------------------*)
process SupRecResponse[c_u,c_l](ver:ccr_ver,aet_self,aet_peer:ae_title,
                                c_br,n_br:atomic_action_branch) : exit :=
        ( c_u?sp:CSP[is_CRecoverReq(sp) and (get_rs(sp) eq commit)];
          ( [c_br eq get_aab(sp) ] ->
            ( c_l!recover_trans_sp_to_pdu(sp,ver,aet_self,aet_peer);
              SupRecActions[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
        ) ) )
        []
        ( c_u?sp:CSP[is_CRecoverRsp(sp) and (get_rr(sp) eq retry)];
          c_l!recover_trans_sp_to_pdu(sp,ver,aet_self,aet_peer);
          i;    (* i works as a retry timer *)
          SupRecovery[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
        )
        []
        ( c_u?sp:CSP[is_CRecoverRsp(sp) and (get_rr(sp) eq unknown)];
          c_l!recover_trans_sp_to_pdu(sp,ver,aet_self,aet_peer);
          exit
        )
endproc (* SupRecResponse *)

(*-------------------------------------------------------------*)
(*      Process for Superior CCR Recovery : Recover            *)
(*      C-RECOVER-RC                                           *)
(*-------------------------------------------------------------*)
process SupRecActions[c_u,c_l](ver:ccr_ver,aet_self,aet_peer:ae_title,
                                c_br,n_br:atomic_action_branch) : exit :=
        ( c_l?pdu:CPDU[is_c_recover_rc(pdu) and (get_rr(pdu) eq done)];
          c_u!recover_trans_pdu_to_sp(pdu,ver,aet_self,aet_peer);
          exit
        )
        []
        ( c_l?pdu:CPDU[is_c_recover_rc(pdu) and (get_rr(pdu) eq retry)];
          c_u!recover_trans_pdu_to_sp(pdu,ver,aet_self,aet_peer);
          SupRecovery[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
        )
endproc (* SupRecActions *)

endproc (* SupRecovery *)

endproc (* SupCCEP *)

(*-------------------------------------------------------------*)
(*      Process for Subordinate CCR End Point                  *)
(*-------------------------------------------------------------*)
process SubCCR[c_u,c_l](ver:ccr_ver,aet_self,aet_peer:AE_title) : exit :=
        SubAction[c_u,c_l](ver,aet_self,aet_peer,aab_null,aab_null)
        []
        SubRecovery[c_u,c_l](ver,aet_self,aet_peer,aab_null,aab_null)
where

(*-------------------------------------------------------------*)
(*      Process for Subordinate CCR Action : Normal            *)
(*      C-BEGIN-RI                                             *)
(*-------------------------------------------------------------*)
process SubAction[c_u,c_l](ver:ccr_ver,aet_self,aet_peer:ae_title,
                                c_br,n_br:atomic_action_branch) : exit :=
        ( c_l?pdu:CPDU[is_c_begin_ri(pdu)];
          c_u!begin_trans_pdu_to_sp(pdu,ver,aet_self,aet_peer);
          ContSubAction[c_u,c_l](ver,aet_self,aet_peer,
                        get_aab(pdu,ver,aet_self,aet_peer),n_br)
        )
where

(*-------------------------------------------------------------*)
(*      Process for Subordinate CCR Action : Normal             *)
(*      C-BEGIN rsp ..... C-PREPARE-RI                          *)
(*-------------------------------------------------------------*)
process ContSubAction[c_u,c_l](ver:ccr_ver,aet_self,aet_peer:ae_title,
                                c_br,n_br:atomic_action_branch) : exit :=
        ( c_u?sp:CSP[is_CBeginRsp(sp)];
          c_l!c_begin_rc(to_pdu(get_ud(sp)));
          ( ( c_l?pdu:CPDU[is_c_prepare_ri(pdu)];
              c_u!CPrepareInd(to_sp(get_ud(pdu)));
              ContSubAction1[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
            )
            []
            ( c_u?sp:CSP[is_CReadyReq(sp)];
              c_l!c_ready_ri(to_pdu(get_ud(sp)));
              ContSubAction2[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
            )
            []
            SubRollBack[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
        ) )
        []
        ( c_l?pdu:CPDU[is_c_prepare_ri(pdu)];
          c_u!CPrepareInd(to_sp(get_ud(pdu)));
          ( ( c_u?sp:CSP[is_CBeginRsp(sp)];
              c_l!c_begin_rc(to_pdu(get_ud(sp)));
              ContSubAction1[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
            )
            []
            ( c_u?sp:CSP[is_CReadyReq(sp)];
              c_l!c_ready_ri(get_ud(pdu));
              ContSubAction3[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
            )
            []
            SubRollBack[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
        ) )
        []
        ( c_u?sp:CSP[is_CReadyReq(sp)];
          c_l!c_ready_ri(to_pdu(get_ud(sp)));
          ContSubAction2[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
        )
        []
        SubRollBack[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
where

(*-------------------------------------------------------------*)
(*      Process for Subordinate CCR Action : Normal             *)
(*      After C-BEGIN rsp and C-PREPARE ind                    *)
(*-------------------------------------------------------------*)
process ContSubAction1[c_u,c_l](ver:ccr_ver,aet_self,aet_peer:ae_title,
                                c_br,n_br:atomic_action_branch) : exit :=
        ( c_u?sp:CSP[is_CReadyReq(sp)];
          c_l!c_ready_ri(to_pdu(get_ud(sp)));
          ContSubAction3[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
        )
        []
        SubRollBack[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
endproc (* ContSubAction1 *)

(*-------------------------------------------------------------*)
(*      Process for Supbordinate CCR Action : Normal           *)
(*      After C-READY req and not C-PREPARE ind                *)
(*-------------------------------------------------------------*)
process ContSubAction2[c_u,c_l](ver:ccr_ver,aet_self,aet_peer:ae_title,
                                c_br,n_br:atomic_action_branch) : exit :=
        ( c_l?pdu:CPDU[is_c_prepare_ri(pdu)];
          c_u!CPrepareInd(to_sp(get_ud(pdu)));
          ContSubAction3[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
        )
        []
        ContSubAction3[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
endproc (* ContSubAction2 *)

(*-------------------------------------------------------------*)
(*      Process for Subordinate CCR Action : Normal            *)
(*      After C-READY req                                      *)
(*-------------------------------------------------------------*)
process ContSubAction3[c_u,c_l](ver:ccr_ver,aet_self,aet_peer:ae_title,
                                c_br,n_br:atomic_action_branch) : exit :=
        ( c_l?pdu:CPDU[is_c_commit_ri(pdu)];
          c_u!CCommitInd(to_sp(get_ud(pdu)));
          c_u?sp:CSP[is_CCommitRsp(sp)];
          c_l!c_commit_rc(to_pdu(get_ud(sp)));
          exit
        )
        []
        ( c_l?pdu:CPDU[is_c_rollback_ri(pdu)];
          c_u!CRollbackInd(to_sp(get_ud(pdu)));
          c_u?sp:CSP[is_CRollbackRsp(sp)];
          c_l!c_rollback_rc(to_pdu(get_ud(sp)));
          exit
        )
        []
        ( c_l?pdu,pdu1:CPDU[is_c_commit_ri(pdu) and is_c_begin_ri(pdu1)];
          c_u!CCommitInd(to_sp(get_ud(pdu)));
          c_u!begin_trans_pdu_to_sp(pdu1,ver,aet_self,aet_peer);
          c_u?sp:CSP[is_CCommitRsp(sp)];
          c_l!c_commit_rc(to_pdu(get_ud(sp)));
          ContSubAction[c_u,c_l](ver,aet_self,aet_peer,
                       get_aab(pdu,ver,aet_self,aet_peer),aab_null)
        )
        []
        ( c_l?pdu,pdu1:CPDU[is_c_rollback_ri(pdu) and is_c_begin_ri(pdu1)];
          c_u!CRollbackInd(to_sp(get_ud(pdu)));
          c_u!begin_trans_pdu_to_sp(pdu1,ver,aet_self,aet_peer);
          ContSubAction4[c_u,c_l](ver,aet_self,aet_peer,
                           c_br, get_aab(pdu1,ver,aet_self,aet_peer))
        )
endproc (* ContSubAction3 *)

(*------------------------------------------------------------*)
(*      Process for Subordinate CCR Action : Normal           *)
(*      C-ROLLBACK Sequence                                   *)
(*------------------------------------------------------------*)
process SubRollBack[c_u,c_l](ver:ccr_ver,aet_self,aet_peer:ae_title,
                                c_br,n_br:atomic_action_branch) : exit :=
        ( c_l?pdu:CPDU[is_c_rollback_ri(pdu)];
          c_u!CRollbackInd(to_sp(get_ud(pdu)));
          c_u?sp:CSP[is_CRollbackRsp(sp)];
          c_l!c_rollback_rc(to_pdu(get_ud(sp)));
          exit
        )
        []
        ( c_u?sp:CSP[is_CRollbackReq(sp)];
          c_l!c_rollback_ri(to_pdu(get_ud(sp)));
          ( ( c_l?pdu:CPDU[is_c_rollback_ri(pdu)];
              c_u!CRollbackInd(to_sp(get_ud(pdu)));
              c_u?sp:CSP[is_CRollbackRsp(sp)];
              c_l!c_rollback_rc(to_pdu(get_ud(sp)));
              exit
            )
            []
            ( c_l?pdu:CPDU[is_c_rollback_rc(pdu)];
              c_u!CRollbackCnf(to_sp(get_ud(pdu)));
              exit
            )
            []
            ( c_l?pdu,pdu1:CPDU[is_c_rollback_ri(pdu) and is_c_begin_ri(pdu1)];
              c_u!CRollbackInd(to_sp(get_ud(pdu)));
              c_u!begin_trans_pdu_to_sp(pdu1,ver,aet_self,aet_peer);
              ContSubAction4[c_u,c_l](ver,aet_self,aet_peer,
                        c_br, get_aab(pdu1,ver,aet_self,aet_peer))
        ) ) )
        []
        ( c_l?pdu,pdu1:CPDU[is_c_rollback_ri(pdu) and is_c_begin_ri(pdu1)];
          c_u!CRollbackInd(to_sp(get_ud(pdu)));
          c_u!begin_trans_pdu_to_sp(pdu1,ver,aet_self,aet_peer);
          ContSubAction4[c_u,c_l](ver,aet_self,aet_peer,
                        c_br, get_aab(pdu1,ver,aet_self,aet_peer))
        )
endproc (* SubRollBack *)

(*------------------------------------------------------------*)
(*      Process for Subordinate CCR Action : Normal           *)
(*      Wait for C-ROLLBACK-RC                                *)
```

```
(*--------------------------------------------------------------*)
process ContSubAction4[c_u,c_l](ver:ccr_ver,aet_self,aet_peer:ae_title,
                                c_br,n_br:atomic_action_branch) : exit :=
        ( c_u?sp:CSP[is_CRollbackRsp(sp)];
          c_l!c_rollback_rc(to_pdu(get_ud(sp)));
          ContSubAction[c_u,c_l](ver,aet_self,aet_peer,n_br,aab_null)
        )
endproc (* ContSubAction4 *)

endproc (* ContSubAction *)

endproc (* SubAction *)

(*--------------------------------------------------------------*)
(*      Process for Subordinate CCR Action : Recover            *)
(*      C-RECOVER req or C-PREPARE-RI                           *)
(*--------------------------------------------------------------*)
process SubRecovery[c_u,c_l](ver:ccr_ver,aet_self,aet_peer:ae_title,
                                c_br,n_br:atomic_action_branch) : exit :=
        ( c_u?sp:CSP[is_CRecoverReq(sp) and (get_rs(sp) eq ready)];
          c_l!recover_trans_sp_to_pdu(sp,ver,aet_self,aet_peer);
          SubRecResponse[c_u,c_l](ver,aet_self,aet_peer,get_aab(sp),n_br)
        )
        []
        ( c_l?pdu:CPDU[is_c_recover_ri(pdu) and (get_rs(pdu) eq commit)];
          c_u!recover_trans_pdu_to_sp(pdu,ver,aet_self,aet_peer);
          SubRecActions[c_u,c_l](ver,aet_self,aet_peer,
                        get_aab(pdu,ver,aet_self,aet_peer),n_br)
        )
where

(*--------------------------------------------------------------*)
(*      Process for Subordinate CCR Action : Recover            *)
(*      C-RECOVER req or C-RECOVER-RI                           *)
(*--------------------------------------------------------------*)
process SubRecResponse[c_u,c_l](ver:ccr_ver,aet_self,aet_peer:AE_title,
                                c_br,n_br:atomic_action_branch) : exit :=
        ( c_l?pdu:CPDU[is_c_recover_rc(pdu) and (get_rr(pdu) eq unknown)];
          c_u!recover_trans_pdu_to_sp(pdu,ver,aet_self,aet_peer);
          exit
        )
        []
        ( c_l?pdu:CPDU[is_c_recover_rc(pdu) and (get_rr(pdu) eq retry)];
          c_u!recover_trans_pdu_to_sp(pdu,ver,aet_self,aet_peer);
          SubRecovery[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
        )
        []
        ( c_l?pdu:CPDU[is_c_recover_ri(pdu) and (get_rs(pdu) eq commit)];
          c_u!recover_trans_pdu_to_sp(pdu,ver,aet_self,aet_peer);
          SubRecActions[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
        )
endproc (* SubRecResponse *)

(*--------------------------------------------------------------*)
```

```
(*      Process for Subordinate CCR Action : Recover           *)
(*      C-RECOVER-RC                                           *)
(*-------------------------------------------------------------*)
process SubRecActions[c_u,c_l](ver:ccr_ver,aet_self,aet_peer:AE_title,
                                c_br,n_br:atomic_action_branch) : exit :=
        ( c_u?sp:CSP[is_CRecoverRsp(sp) and (get_rr(sp) eq done)];
          c_l!recover_trans_sp_to_pdu(sp,ver,aet_self,aet_peer);
          exit
        )
        []
        ( c_u?sp:CSP[is_CRecoverRsp(sp)and (get_rr(sp) eq retry)];
          c_l!recover_trans_sp_to_pdu(sp,ver,aet_self,aet_peer);
          SubRecovery[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
        )
endproc (* SubRecActions *)

endproc (* SubRecovery *)

endproc (* SubCCR *)

endproc (* CCRPM *)

endproc (* CCR *)

(*-------------------------------------------------------------*)
(*      Control Function :                                     *)
(*      In CF (Control Function) the following three processes *)
(*      are defined;                                           *)
(*      CCRRouter :                                            *)
(*              to indicate the routing of Applcation Layer    *)
(*              PDUs. Especially the case od c-initialise is   *)
(*              imported.                                      *)
(*      ConcDeconcProc :                                       *)
(*              Concatenator and Deconcatenator of Application *)
(*              Layer PDUs.                                    *)
(*      LowerMapper :                                          *)
(*              to indicate how to map the Application Layer's *)
(*              PDUs to the Presentation Layer Service         *)
(*              Primitives.                                    *)
(*-------------------------------------------------------------*)
process CF[u,p,c_u,c_l,acse_u,acse_l,env,u_env] : exit :=
        ( hide cd_u,cd_l,l_u,l_l,ver in
          ( CFRouter[u,c_u,c_l,acse_u,acse_l,env,cd_u,cd_l,ver]
            |[cd_u,cd_l,ver]|
            ( ( ConcDeconcProc[cd_u,cd_l,l_u,l_l](q_empty) [> exit )
              |[l_u,l_l]|
              LowerMapper[l_u,l_l,p,u_env,ver](rejected,false)
        ) ) ) [> env!failure;exit
where

(*-------------------------------------------------------------*)
(*      CF Router :                                            *)
(*      In this description, CCR part of the routing function  *)
(*      is only important.                                     *)
```

```
(*-------------------------------------------------------------*)
process CFRouter[u,c_u,c_l,acse_u,acse_l,env,cd_u,cd_l,ver] : exit :=
        CCRPart[u,c_u,c_l,acse_u,acse_l,env,cd_u,cd_l,ver]
(*
        |[acse_u,acse_l]|
        ACSEPart[u,acse_u,acse_l,cd_u,cd_l]
        |||
        OtherPart[u,cd_u,cd_l]

        CFRouter also should have ACSEPart and Other parts, but these parts
        are not important in CCR Specification. Therefore, in this
        specification they are not described.
*)
where

(*-------------------------------------------------------------*)
(*      CCR Part in CF :                                       *)
(*      NonNegotiateCCRInitialize works if CCR version number   *)
(*      is fixed as 1. In this case the negotiation is not      *)
(*      necessary.                                             *)
(*      NegotiateCCRInitialize is reponsible for the negotiation*)
(*       of CCR protocol version. After fixing the version,     *)
(*      if it succeeds, CCRLoop works.                         *)
(*-------------------------------------------------------------*)
process CCRPart[u,c_u,c_l,acse_u,acse_l,env,cd_u,cd_l,ver] : exit :=
        ( c_u?ver:ccr_ver;
          ( [ver eq ver1]->
                NonNegotiateCCRPartInitialize[u,c_u,c_l,
                                acse_u,acse_l,cd_u,cd_l]
            []
            [ver contains ver2]->
                NegotiateCCRPartInitialize[u,c_u,c_l,acse_u,acse_l,cd_u,cd_l]
          )
          []
          env?ver:ccr_ver?aet_self:AE_title?aet_peer:AE_title;
          exit(ver)
        ) >> accept ver:ccr_ver in
                ( ver ! ver;
                  ( [ver eq rejected]->exit
                    []
                    [ver ne rejected]->CCRLoop[u,c_u,c_l,cd_u,cd_l]
                ) )
where

(*-------------------------------------------------------------*)
(*      CCR Initializer in CF for Non-Negotiated Version Case : *)
(*      CCR version is fixed as version 1.                     *)
(*-------------------------------------------------------------*)
process NonNegotiateCCRPartInitialize[u,c_u,c_l,acse_u,acse_l,cd_u,cd_l]
                                                        : exit(ccr_ver) :=
        NonNegotiateCCRPartInitiator[u,c_u,c_l,acse_u,acse_l,cd_u,cd_l]
        []
        NonNegotiateCCRPartResponder[u,c_u,c_l,acse_u,acse_l,cd_u,cd_l]
where
```

```
(*-------------------------------------------------------------*)
(*      CCR Initializer in CF for Non-Negotiated Version Case : *)
(*      Initiator                                               *)
(*      CCR version is fxed as version 1.                       *)
(*-------------------------------------------------------------*)
process NonNegotiateCCRPartInitiator[u,c_u,c_l,acse_u,acse_l,cd_u,cd_l]
                                                        : exit(ccr_ver) :=
        ( u?sp:SP[is_acse_sp(sp) and is_AAssocReq(get_asp(sp))];
          ( let asp1:ASP = get_asp(sp) in
              acse_u!asp1;
              acse_l?psp:PSP;
                (* This psp is expected to be P-CONNECT req.
                   The transformation from ASP to PSP is carried  out
                   in ACSE Protocol Machine, which is omitted in
                   this description.
                *)
              cd_u!psp;
              cd_l?psp:PSP;
                (* This psp is expected to be P-CONNECT cnf.
                   The check is carried out in ACSE Protocol Machine,
                   which is omitted in this description.
                *)
              acse_l!psp;
              acse_u?asp2:ASP[is_AAssocCnf(asp2)];
              u!sp(asp2);
              ( [(get_rat(asp2) eq ap_null) or (get_raq(asp2) eq aq_null)]->
                        c_u!aet(get_cgat(asp1),get_cgaq(asp1))
                           !aet(get_cdat(asp1),get_cdaq(asp1));
                        exit(ver1)
                []
                [(get_rat(asp2) ne ap_null) and (get_raq(asp2) ne aq_null)]->
                        c_u!aet(get_cgat(asp1),get_cgaq(asp1))
                           !aet(get_rat(asp2),get_raq(asp2));
                        exit(ver1)
        ) ) )
endproc (* NonNegotiateCCRPartInitiator *)

(*-------------------------------------------------------------*)
(*      CCR Initializor in CF for Negotiated Version Case :     *)
(*      Responder                                               *)
(*      CCR version is fxed as version 1.                       *)
(*-------------------------------------------------------------*)
process NonNegotiateCCRPartResponder[u,c_u,c_l,acse_u,acse_l,cd_u,cd_l]
                                                        : exit(ccr_ver) :=
        ( cd_l?psp:PSP;
                (* This psp is expected to be P-CONNECT ind.
                   The check is carried out in ACSE Protocol
                   Machine, which is omitted in this description.
                *)
          acse_l!psp;
          acse_u?asp1:ASP[is_AAssocInd(asp1)];
          ( u!sp(asp1);
            u?sp:SP[is_acse_sp(sp) and is_AAssocRsp(get_asp(sp))];
            ( let asp2:ASP = get_asp(sp) in
              acse_u!asp2;
              acse_l?psp:PSP;
                (* This psp is expected to be P-CONNECT rsp.
                   The transformation from ASP to PSP is carried out in
                   ACSE Protocol Machine, which is omitted in this description.
                *)
              cd_u!psp;
              ( [(get_rat(asp2) eq ap_null)
                                or (get_raq(asp2) eq aq_null)]->
                        c_u!aet(get_cdat(asp1),get_cdaq(asp1))
                           !aet(get_cgat(asp1),get_cgaq(asp1));
                        exit(ver1)
                []
                [(get_rat(asp2) ne ap_null)
                                and (get_raq(asp2) ne aq_null)]->
                        c_u!aet(get_rat(asp2),get_raq(asp2))
                           !aet(get_cgat(asp1),get_cgaq(asp1));
                        exit(ver1)
        ) ) ) )
endproc (* NonNegotiateCCRPartResponder *)

endproc (* NonNegotiateCCRPartInitialize *)

(*------------------------------------------------------------*)
(*      CCR Initializer in CF for Negotiated Version Case :     *)
(*      The negotiation of CCR protocol version has two cases;  *)
(*      CCRInitiator :                                          *)
(*              The local CCR provider issues c_initialize_ri   *)
(*              which is the user data of A-ASSOCIATE request   *)
(*              and receives c-initialize-rc which is the user  *)
(*              data of A-ASSOCIATE confirmation.               *)
(*      CCRResponder :                                          *)
(*              the remote CCR provider receives                *)
(*              c-initialize-ri which is the user data of       *)
(*              A-ASSOCIATE indication and issues               *)
(*              c-initialize-rc which is the user data of       *)
(*              A-ASSOCIATE response.                           *)
(*------------------------------------------------------------*)
process NegotiateCCRPartInitialize[u,c_u,c_l,acse_u,acse_l,cd_u,cd_l]
                                                     : exit(ccr_ver) :=
        NegotiateCCRPartInitiator[u,c_u,c_l,acse_u,acse_l,cd_u,cd_l]
        []
        NegotiateCCRPartResponder[u,c_u,c_l,acse_u,acse_l,cd_u,cd_l]
where

(*------------------------------------------------------------*)
(*      CCR Initializer in CF for Negotiated Version Case :     *)
(*      Initiator                                               *)
(*------------------------------------------------------------*)
process NegotiateCCRPartInitiator[u,c_u,c_l,acse_u,acse_l,cd_u,cd_l]
                                                     : exit(ccr_ver) :=
        ( u?sp:SP[is_acse_sp(sp) and is_AAssocReq(get_asp(sp))];
          ( let asp1:ASP = get_asp(sp) in
            ( c_u!AAssoc_req;
              c_l?cpdu:CPDU[is_c_initialize_ri(cpdu)];
              acse_u!AAssocReq(get_cgat(asp1),get_cgaq(asp1),
                               get_cdat(asp1),get_cdaq(asp1),
                               get_ud(asp1)+pdu(cpdu),get_op(asp1));
              acse_l?psp:PSP;
                (* This psp is expected to be P-CONNECT req.
                   The transformation from ASP to PSP is carried out in
                   ACSE Protocol Machine, which is omitted in this description.
                *)
              cd_u!psp;
              cd_l?psp:PSP;
                (* This psp is expected to be P-CONNECT cnf.
                   The check is carried out in ACSE Protocol Machine,
                   which is omitted in this description.
                *)
              acse_l!psp;
              acse_u?asp2:ASP[is_AAssocCnf(asp2)];
              c_l!ccr_pdu(get_ud(asp2));
              c_u?ver:ccr_ver;
              u!sp(AAssocCnf(get_rat(asp2),get_raq(asp2),get_r(asp2),
                             get_rs(asp2),get_d(asp2),ase_pdu(get_ud(asp2)),
                             get_op(asp2)));
              ( [(get_rat(asp2) eq ap_null) or (get_raq(asp2) eq aq_null)]->
                        c_u!aet(get_cgat(asp1),get_cgaq(asp1))
                           !aet(get_cdat(asp1),get_cdaq(asp1));
                        exit(ver)
                []
                [(get_rat(asp2) ne ap_null) and (get_raq(asp2) ne aq_null)]->
                        c_u!aet(get_cgat(asp1),get_cgaq(asp1))
                           !aet(get_rat(asp2),get_raq(asp2));
                        exit(ver)
        ) ) ) )
endproc (* NegoatiateCCRPartInitiator *)

(*-------------------------------------------------------------*)
(*      CCR Initializor in CF for Negotiated Version Case :     *)
(*      Responder                                               *)
(*-------------------------------------------------------------*)
process NegotiateCCRPartResponder[u,c_u,c_l,acse_u,acse_l,cd_u,cd_l]
                                                        : exit(ccr_ver) :=
        ( cd_l?psp:PSP;
                (* This psp is expected to be P-CONNECT ind.
                   The check is carried out in ACSE Protocol
                   Machine, which is omitted in this description.
                *)
          acse_l!psp;
          acse_u?asp1:ASP[is_AAssocInd(asp1)];
          c_l!ccr_pdu(get_ud(asp1));
          c_u?ver:ccr_ver;
          c_u!ver;
          ( [ver eq rejected]->
            ( acse_u!AAssocRsp(get_cdat(asp1),get_cdaq(asp1),rejected_permanent,
                               ACSE_service_user,no_reason_given,q_empty,
                          get_op(asp1));
              acse_l?psp:PSP;
              cd_u!psp;
              exit(rejected)
            )
            []
            [ ver ne rejected ] ->
            ( u!sp(AAssocInd(get_cgat(asp1),get_cgaq(asp1),
                        get_cdat(asp1),get_cdaq(asp1),
                        ase_pdu(get_ud(asp1)),get_op(asp1)));
              u?sp:SP[is_acse_sp(sp) and is_AAssocRsp(get_asp(sp))];
              ( let asp2:ASP = get_asp(sp) in
                ( c_u!AAssoc_rsp;
                  c_l?cpdu:CPDU;
                  acse_u!AAssocRsp(get_rat(asp2),get_raq(asp2),get_r(asp2),
                        get_rs(asp2),get_d(asp2),
                        get_ud(asp2)+pdu(cpdu),
                        get_op(asp2));
                  acse_l?psp:PSP;
                        (* This psp is expected to be P-CONNECT rsp.
                           The transformation from ASP to PSP is
                           carried out in ACSE Protocol Machine, which
                           is omitted in this description.
                        *)
                  cd_u!psp;
                  ( [(get_rat(asp2) eq ap_null)
                                    or (get_raq(asp2) eq aq_null)]->
                        c_u!aet(get_cdat(asp1),get_cdaq(asp1))
                           !aet(get_cgat(asp1),get_cgaq(asp1));
                        exit(ver)
                    []
                    [(get_rat(asp2) ne ap_null)
                                    and (get_raq(asp2) ne aq_null)]->
                        c_u!aet(get_rat(asp2),get_raq(asp2))
                           !aet(get_cgat(asp1),get_cgaq(asp1));
                        exit(ver)
        ) ) ) ) ) )
endproc (* NegotiateCCRPartResponder *)

endproc (* NegotiateCCRPartInitialize *)

(*-------------------------------------------------------------*)
(*      CCR Loop in CF :                                       *)
(*      Normal case of CCR PDU routing after the establishmment *)
(*      of an association.                                     *)
(*-------------------------------------------------------------*)
process CCRLoop[u,c_u,c_l,cd_u,cd_l] : noexit :=
        ( ( u?sp:SP[is_ccr_sp(sp)];
            ( c_u!get_csp(sp);
              ( c_l?cpdu:CPDU;
                cd_u!pdu(cpdu);
                exit
                []
                (* c_commit_ri + c_begin_ri and c_rollback_ri + c_begin_ri *)
                c_l?cpdu:CPDU?cpdu1:CPDU;
                cd_u!pdu(cpdu)!pdu(cpdu1);
                exit
          ) ) )
          []
          ( cd_l?pdu:PDU[is_ccr_pdu(pdu)];
            ( c_l!get_cpdu(pdu);
              c_u?csp:CSP;
              u!sp(csp);
              exit
          ) )
          []
          (* c_commit_ri + c_begin_ri and c_rollback_ri + c_begin_ri  *)
          (* In this case two PDUs simultaneouly appear at cd_l gate. *)
          ( cd_l?pdu,pdu1:PDU[is_ccr_pdu(pdu) and is_ccr_pdu(pdu1)];
            ( c_l!get_cpdu(pdu)!get_cpdu(pdu1);
              c_u?csp:CSP;
              u!sp(csp);
              c_u?csp:CSP;
              u!sp(csp);
              exit
        ) ) )
        >> CCRLoop[u,c_u,c_l,cd_u,cd_l]
endproc (* CCRLoop *)

endproc (* CCRPart *)

endproc (* CFRouter *)

(*-------------------------------------------------------------*)
(*      Concatenator and Deconcatenator Process :              *)
(*      CCR PDUs and other Application Layer PDUs are to be    *)
(*      concatenated and deconcatenated as the description     *)
(*      which is subject to CCR protocol document, ISO/IEC     *)
(*      IS 9805 chapter 10.                                    *)
(*-------------------------------------------------------------*)
process ConcDeconcProc[cd_u,cd_l,l_u,l_l](pdu_q:PDUqueue) : noexit :=
        Concatenator[cd_u,l_u](pdu_q)
        |||
        Deconcatenator[cd_l,l_l]
where

(*-------------------------------------------------------------*)
(*      Concatenator Process :                                 *)
(*      This process (Concatenator) and the next one (Concat)  *)
(*      are responsible for the concatenation of PDUs.         *)
(*      Especially, in Concat, the operator "may_conc" is      *)
(*      essential, which checks whether incoming PDUs can be   *)
(*      concatenated or not.                                   *)
(*-------------------------------------------------------------*)
process Concatenator[cd_u,l_u](pdu_q:PDUqueue) : noexit :=
        cd_u?pdu:PDU;Concat[cd_u,l_u](pdu_q,pdu,empty)
        []
        cd_u?pdu:PDU?pdu1:PDU;Concat[cd_u,l_u](pdu_q,pdu,pdu1)
        []
        [not(is_q_empty(pdu_q))]->
        (         i;    (* The concatenated PDU can be issued in any time. *)
          l_u!pdu_q;
          Concatenator[cd_u,l_u](q_empty of PDUqueue)
        )
        [] (* Presentation Service primtitives may happen at the gate
                   cd_u. It is passed through to the LowerMapper process.
           *)
        ( cd_u?psp:PSP;
          ( [is_q_empty(pdu_q)]->
                l_u!psp;
                Concatenator[cd_u,l_u](q_empty)
            []
            [not(is_q_empty(pdu_q))]->
                l_u!pdu_q;l_u!psp;
                Concatenator[cd_u,l_u](q_empty)
        ) )
where

(*-------------------------------------------------------------*)
(*      Sub-Concatenator Process :                             *)
(*      In the case of c-commit-ri + c-begib-ri and            *)
(*      c-rollback-ri + c-begin-ri, which is represented by    *)
(*      that pdu1 is not empty, two PDUs which are "c-commit-ri *)
(*      and c-begin-ri" or "c-rollback-ri and c-begin-ri" are   *)
(*      forced to be concatenated.                             *)
(*-------------------------------------------------------------*)
process Concat[cd_u,l_u](pdu_q:PDUqueue,pdu:PDU,pdu1:PDU) : noexit :=
        [not(is_q_empty(pdu_q))]->
        (         i;
          l_u!pdu_q;
          Concatenator[cd_u,l_u](queue(pdu))
        )
        []
        [is_empty(pdu1)]->
        ( [may_conc(h(pdu_q),h(pdu))]->
                Concatenator[cd_u,l_u](pdu_q + pdu)
          []
          [not(may_conc(h(pdu_q),h(pdu)))]->
          ( l_u!pdu_q;
            Concatenator[cd_u,l_u](queue(pdu))
        ) )
        []
        [not(is_empty(pdu1))]->
        (* c_commit_ri + c_begin_ri and c_rollback_ri + c_begin_ri *)
        ( [is_q_empty(pdu_q)]->
                l_u!(queue(pdu) + pdu1);
                Concatenator[cd_u,l_u](q_empty)
          []
          [not(is_q_empty(pdu_q))]->
                l_u!pdu_q;
                l_u!(queue(pdu) + pdu1);
                Concatenator[cd_u,l_u](q_empty)
        )
endproc (* Concat *)

endproc (* Concatenator *)

(*-------------------------------------------------------------*)
(*      Deconcatenator Process :                               *)
(*      This process (Deconcatenator) ans the next one         *)
(*      (Deconcat) are responsible for the deconcatenation of  *)
(*      Application Layer PDUs.                                *)
(*-------------------------------------------------------------*)
process Deconcatenator[cd_l,l_l] : noexit :=
        ( l_l?pdu_q:PDUqueue;
          ( Deconcat[cd_l](pdu_q) >> Deconcatenator[cd_l,l_l] )
        )
        []
        ( l_l?psp:PSP;cd_l!psp;Deconcatenator[cd_l,l_l]
        )
where

(*-------------------------------------------------------------*)
(*      Sub-Deconcatenator Process :                           *)
(*      In the case of c-commit-ri + c-begib-ri and            *)
(*      c-rollback-ri + c-begin-ri, which is represented by    *)
(*      that pdu1 is not empty, two PDUs which are "c-commit-ri *)
(*      and c-begin-ri" or "c-rollback-ri and c-begin-ri" are   *)
(*      issued simultaneously at cd_l gate.                    *)
(*-------------------------------------------------------------*)
process Deconcat[cd_l](pdu_q:PDUqueue) : exit :=
        [not(is_q_empty(pdu_q))]->
        ( [is_c_commit_ri_seq(h(pdu_q)) or is_c_rollback_ri_seq(h(pdu_q))]->
          (* c_commit_ri + c_begin_ri and c_rollback_ri + c_begin_ri *)
                cd_l!head(pdu_q)!head(tail(pdu_q));
                Deconcat[cd_l](tail(tail(pdu_q)))
          []
          [not(is_c_commit_ri_seq(h(pdu_q))
                                or is_c_rollback_ri_seq(h(pdu_q)))]->
                cd_l!head(pdu_q);Deconcat[cd_l](tail(pdu_q))
        )
        []
        [is_q_empty(pdu_q)]->exit
endproc (* Deconcat *)

endproc (* Deconcatenator *)

endproc (* ConcDeconcProc *)

(*-------------------------------------------------------------*)
(*      Lower Service-PDU Mapping Process :                    *)
(*      The concatenated PDUs should be mapped into Presentation*)
(*      Layer Service Primitives as the description which is   *)
(*      subnitted to CCR protocol documents, ISO/IEC IS 9805   *)
(*      chapter 9 and 10.                                      *)
(*-------------------------------------------------------------*)
```

```
process LowerMapper[l_u,l_l,p,u_env,ver](ver:ccr_ver,c_r_b:Bool) : exit :=
        ( l_u?pdu_q:PDUqueue;
          ( ( (* C-BEGIN-RI -> P-SYNC-MINOR req *)
              [h(ccr_pdu(pdu_q)) eq c_begin_ri ]->
              ( [ver eq ver1]->
                ( p!PSyncMinReq(opt,s_null,pdu_q,off);
                  LowerMapper[l_u,l_l,p,u_env,ver](ver,false)
                )
                []
                [ver eq ver2]->
                (* In version 2, Data Separation Functional Unit is used. *)
                ( p!PSyncMinReq(opt,s_null,pdu_q,on);
                  LowerMapper[l_u,l_l,p,u_env,ver](ver,false)
            ) ) )
            []
            (* Normally, C-BEGIN-RC -> P-SYNC-MINOR req.
               But after C-BEGIN-RI with C-ROLLBACK-RI or C-COMMIT-RI,
               C-BEGIN-RC -> P-TYPED-DATA req. If the latter case happens,
               the variable "c_r_b" is set to true.
            *)
            [h(ccr_pdu(pdu_q)) eq c_begin_rc]->
            ( [c_r_b eq false]->
              ( u_env?s_p:sync_point;
                p!PSyncMinRsp(s_p,pdu_q);
                LowerMapper[l_u,l_l,p,u_env,ver](ver,c_r_b)
              )
              []
              [c_r_b eq true]->
              ( p!PTypedDataReq(pdu_q);
                LowerMapper[l_u,l_l,p,u_env,ver](ver,false)
            ) )
            []
            (* C-PREPARE-RI -> P-TYPED-DATA req *)
            [h(ccr_pdu(pdu_q)) eq c_prepare_ri]->
            ( p!PTypedDataReq(pdu_q);
              LowerMapper[l_u,l_l,p,u_env,ver](ver,c_r_b)
            )
            []
            (* C-READY-RI -> P-TYPED-DATA req *)
            [h(ccr_pdu(pdu_q)) eq c_ready_ri]->
            ( p!PTypedDataReq(pdu_q);
              LowerMapper[l_u,l_l,p,u_env,ver](ver,c_r_b)
            )
            []
            (* C-COMMIT-RI or C-COMMIT-RI + C-BEGIN-RI case *)
            [h(ccr_pdu(pdu_q)) eq c_commit_ri]->
            ( [ver eq ver1]-> (* If version is 1, mapped to P-SYNC-MAJOR req *)
              ( p!PSyncMajReq(s_null,pdu_q);
                LowerMapper[l_u,l_l,p,u_env,ver](ver,true)
              )
              []
              [ver eq ver2]-> (* If version is 2, mapped to P-SYNC-MINOR req
                                 with data separation ON *)
              ( p!PSyncMinReq(exp,s_null,pdu_q,on);
    LowerMapper[l_u,l_l,p,u_env,ver](ver,true)
) )
[]
(* C-COMMIT-RC case *)
[h(ccr_pdu(pdu_q)) eq c_commit_rc]->
( [ver eq ver1]-> (* If version is 1, mapped to P-SYNC-MAJOR rsp *)
  ( p!PSyncMajRsp(pdu_q);
    LowerMapper[l_u,l_l,p,u_env,ver](ver,c_r_b)
  )
  []
  [ver eq ver2]-> (* If version is 2, mapped to P-SYNC-MINOR rsp
                   with data separation ON *)
  ( u_env?s_p:sync_point;
    p!PSyncMinRsp(s_p,pdu_q);
    LowerMapper[l_u,l_l,p,u_env,ver](ver,c_r_b)
) )
[]
(* C-ROLLBACK-RI or C-ROLLBACK-RI + C-BEGIN-RI case *)
[h(ccr_pdu(pdu_q)) eq c_rollback_ri]->
( u_env?s_p:sync_point?tk:token;
  ( [ver eq ver1]-> (* If version is 1,
                     mapped to P-RESYNC(restart) req *)
    ( p!PReSyncReq(restart,s_p,tk,pdu_q);
      LowerMapper[l_u,l_l,p,u_env,ver](ver,true)
    )
    []
    [ver eq ver2]-> (* If version is 2,
                     mapped to P-RESYNC(abandon) req *)
    ( p!PReSyncReq(abandon,s_null,tk,pdu_q);
      LowerMapper[l_u,l_l,p,u_env,ver](ver,true)
) ) )
[]
(* C-ROLLBACK-RC case *)
[h(ccr_pdu(pdu_q)) eq c_rollback_rc]->
( u_env?s_p:sync_point;
  ( [ver eq ver1]-> (* If version is 1,
                     mapped to P-RESYNC(restart) rsp *)
    ( p!PReSyncRsp(s_p,t_null,p_null,pdu_q);
      LowerMapper[l_u,l_l,p,u_env,ver](ver,c_r_b)
    )
    []
    [ver eq ver2]-> (* If version is 2,
                     mapped to P-RESYNC(abandon) rsp *)
    ( p!PReSyncRsp(s_null,t_null,p_null,pdu_q);
      LowerMapper[l_u,l_l,p,u_env,ver](ver,c_r_b)
) ) )
[]
(* C-RECOVER-RI -> P-TYPED-DATA req *)
[h(ccr_pdu(pdu_q)) eq c_recover_ri]->
( p!PTypedDataReq(pdu_q);
  LowerMapper[l_u,l_l,p,u_env,ver](ver,c_r_b)
)
[]
(* C-RECOVER-RC -> P-TYPED-DATA req *)
```

```
            [h(ccr_pdu(pdu_q)) eq c_recover_rc]->
            ( p!PTypedDataReq(pdu_q);
              LowerMapper[l_u,l_l,p,u_env,ver](ver,c_r_b)
            )
        ) )
        []
        (* receive Presentation Service Primitives from ASEs
           and issue it to Presentation Layer
        *)
        ( l_u?psp:PSP;
          p!psp;
          LowerMapper[l_u,l_l,p,u_env,ver](ver,c_r_b)
        )
        []
        (* receive Presentation Service Primitives from Presentation Layer
           and issue it to ASEs
        *)
        (* To ACSE *)
        ( p?psp:PSP[is_PSP_for_ACSE(psp)];
          l_l!psp;
          LowerMapper[l_u,l_l,p,u_env,ver](ver,c_r_b)
        )
        []
        (* To other ASEs. In this specification CCR ASE is only considered *)
        ( p?psp:PSP[not(is_PSP_for_ACSE(psp))];
          l_l!get_ud(psp);
          u_env!psp;    (* For some procedures to treat synchronization
                           point etc.
                        *)
          LowerMapper[l_u,l_l,p,u_env,ver](ver,c_r_b)
        )
        []
        (* For the negotiation of CCR protocol version number,
           if rejected, this process exits.
        *)
        ( ver?ver:ccr_ver;
          ( [ver eq rejected]->exit
            []
            [ver ne rejected]->LowerMapper[l_u,l_l,p,u_env,ver](ver,c_r_b)
        ) )
endproc (* LowerMapper *)

endproc (* CF *)

endspec (* ISO_9805_CCR_Spec *)
```

Chapter 3

OSI TP protocol

3.1 Comments

We only examine the data part of the OSI-TP protocol, which is quite large, and we keep the behaviour part unchanged.

3.2 Formal description in both LOTOS and E-LOTOS

```
(*
 *      H.1     Introduction
 *)
(* comment skiped *)
(*
 *      H.1.1   Top Level Specification
 *)
specification OSITP[tpsu, p] : noexit

behaviour
        stop
(* control part skiped *)
where
(*
 *      H.5     Datatyping
 *)
(*
 *      H.5.1   Basic Datatypes
 *)
type Boolean is
sorts bool
opns
  true (*! constructor *), false (*! constructor *) : -> bool
  not : bool -> bool
  _and_, _or_, _implies_ : bool, bool -> bool
eqns
    forall x : bool
  ofsort bool
    not(true) = false;
    not(false) = true;
    true and x = x;
    false and x = false;
    true or x = true;
    false or x = x;
    true implies x = x;
    false implies x = true;
endtype
(* ---------------------------------------------------------------------------
module Boolean will be predefined
========================================================================== *)

type NaturalNumber is Boolean
sorts nat
opns
  0 (*! constructor *), 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 : -> nat
  succ (*! constructor *), decr : nat -> nat
  _eq_, _ne_, _lt_, _le_ : nat, nat -> bool
eqns
    forall m, n : nat
  ofsort nat
    decr(0) = 0;
    decr(succ(n)) = n;
    1 = succ(0);
    2 = succ(1);
    3 = succ(2);
    4 = succ(3);
    5 = succ(4);
    6 = succ(5);
    7 = succ(6);
    8 = succ(7);
    9 = succ(8);
    10 = succ(9);
  ofsort bool
    0 eq 0 = true;
    0 eq succ(m) = false;
    succ(m) eq 0 = false;
    succ(m) eq succ(n) = m eq n;
    m ne n = not(m eq n);
    m lt 0 = false;
    0 lt succ(n) = true;
    succ(m) lt succ(n) = m lt n;
    m le n = not(n lt m);
endtype
(* ---------------------------------------------------------------------------
module NaturalNumber will be predefined
========================================================================== *)

type BitString is
  NaturalNumber
(*
This is not the BitString defined in the standard lotos library,
instead it is a string of boolean, which more closes matches the type
BitString in the ASN.1 definitions of the PDUs etc.
 *)
sorts bit_string
opns
  string (*! constructor *)  : bool -> bit_string
  _+_ (*! constructor *)  : bool, bit_string -> bit_string
  _implies_ : bit_string, bit_string -> bool
  all_false : bit_string -> bool
  get_bit : nat, bit_string -> bool
  set_bit : nat, bool, bit_string -> bit_string
eqns
    forall bs, bs1 : bit_string, b, b1 : bool, x : nat
  ofsort bool
    string(b) implies string(b1) = b implies b1;
    string(b) implies (b1 + bs1) = b implies b1;
    (b + bs) implies string(b1) = (b implies b1) and all_false(bs);
    (b + bs) implies (b1 + bs1) = (b implies b1) and (bs implies bs1);
    all_false(string(b)) = not(b);
    all_false(b + bs) = not(b) and all_false(bs);
    get_bit(0, string(b)) = b;
    get_bit(0, b + bs) = b;
    get_bit(succ(x), string(b)) = false;
    get_bit(succ(x), b + bs) = get_bit(x, bs);
  ofsort bit_string
    set_bit(0, b1, string(b)) = string(b1);
    set_bit(0, b1, b + bs) = b1 + bs;
    set_bit(succ(x), b1, string(b)) = b + set_bit(x, b, string(false));
    set_bit(succ(x), b1, b + bs) = b + set_bit(x, b1, bs);
endtype
(* -----------------------------------------------------------------------
   BitString will be predefined. However, the module used here seems differ
 to standard BitString, and the translation is given.

module BitString is
  NaturalNumber
    type bit_string is
        string (B: bool)
    |   _+_ (B: bool, BS: bit_string)
    endtype

    function all_false (BS1: bit_string) : bool is
        case BS1 in
            string (B:=b1: bool) -> not(b1)
        |   (B:=b1: bool) + (BS:=s1: bit_string) -> not(b1) and all_false(s1)
        endcase
    endfunc

    function _implies_ (BS1: bit_string, BS2: bit_string) : bool is
        case BS1 in
            string (B:=b1: bool) -> b1 implies (select B in BS2)
        |   (B:=b1: bool) + (BS:=s1: bit_string) ->
                case BS2 in
                    string(B:=b2: bool) -> (b1 implies b2) and all_false(s1)
                |   (B:=b2: bool) + (BS:=s2: bit_string) ->
                                (b1 implies b2) and (s1 implies s2)
                endcase
        endcase
    endfunc

    function get_bit (N: nat, BS1: bit_string) : bool is
        case N in
            0 -> select B in BS1
        |   any nat -> get_bit (N-1, select BS in BS1)
        endcase
    endfunc

    function set_bit (N: nat, b: bool, BS1: bit_string) : bit_string is
        case N in
            0 -> case BS1 in
                     string(B:=b1: bool) -> string(b1)
                 |   (B:=b1: bool) + (BS:=s1: bit_string) -> b + s1
                 endcase
        | any nat ->  case BS1 in
                          string(B:=b1: bool) -> b1 + set_bit (N-1, b, string(false))
                      |   (B:=b1: bool) + (BS:=s1: bit_string) -> b1 + set_bit (N-1, b, s1)
                      endcase
        endcase
    endfunc
endmod
=========================================================================== *)

type external_datatypes is Boolean
sorts
  Integer (*! external *),
  OctetString (*! external *),
  AP_title (*! external *),   (*  application process title  *)
  AE_qual (*! external *),    (*  application entity qualifier  *)
  p_address (*! external *),  (*  PSAP, PSEI  *)
  QOS_ISO8326 (*! external *)
opns
  _eq_ (*! external *) : Integer, Integer -> bool
  _eq_ (*! external *) : OctetString, OctetString -> bool
  _eq_ (*! external *) : AP_title, AP_title -> bool
  _eq_ (*! external *) : AE_qual, AE_qual ->  bool
(*  p_address:  PSAP, PSEI  -> p_address  *)
  _<=_ (*! external *) :  QOS_ISO8326, QOS_ISO8326 -> bool
(* qos1 <= qos2 if qos1 supports all the qos requirements specified in qos2 *)
endtype
(* -------------------------------------------------------------------------
    External implementation will be allowed. However, for Integer and
 OctetString efficient built-in types will be provided.
=========================================================================== *)


(*
 *      H.5.2   Parameter Datatypes
 *)
```

```
type TP_parameters is BitString
sorts
  tp_P_abort_diagnostic,
  tp_begin_dialogue_confirmation,
  tp_begin_dialogue_diagnostic_Opt,
  tp_begin_channel_diagnostic_Opt,
  tp_begin_dialogue_result,
  tp_assoc_estab_diagnostic_Opt,
  tp_bid_result,
  channel_utilization,
  detach_type,
  confirmation_urgency,
  confirmation_urgency_Opt,
  contention_winner_assignment,
  tp_defer_type,
  heuristic_report,
  heuristic_report_Opt,
  mapping,
  functional_units,
  functional_units_Opt,
  tp_abort_type,
  reason,
  bool_Opt
opns
(* P_abort_diagnostic *)
  permanent_failure (*! constructor *),
  transient_failure (*! constructor *),
  begin_transaction_reject (*! constructor *),
  end_dialogue_collision (*! constructor *),
  begin_transaction_end_dialogue_collision (*! constructor *),
  protocol_error  (*! constructor *) : -> tp_P_abort_diagnostic
  is_begin_trans_rej : tp_P_abort_diagnostic -> bool

(* begin_dialogue_confirmation *)
  always (*! constructor *), negative (*! constructor *) : -> tp_begin_dialogue_confirmation
  is_always, is_negative : tp_begin_dialogue_confirmation -> bool

(* begin_dialogue_diagnostic *)
  recipient_unknown (*! constructor *),
  recipient_tpsu_unknown (*! constructor *),
  tpsu_not_available_permanent (*! constructor *),
  tpsu_not_available_transient (*! constructor *),
  recipient_tpsu_title_required (*! constructor *),
  functional_unit_not_supported (*! constructor *),
  functional_unit_combination_not_supported (*! constructor *),
  association_reserved (*! constructor *),
  no_reason_given (*! constructor *),
  bdd_absent (*! constructor *) :
    -> tp_begin_dialogue_diagnostic_Opt
  dial_nat :
    tp_begin_dialogue_diagnostic_Opt -> nat
  is_functional_unit_not_supported, is_association_reserved :
    tp_begin_dialogue_diagnostic_Opt -> bool
  suppress_assoc_resv :
    tp_begin_dialogue_diagnostic_Opt -> tp_begin_dialogue_diagnostic_Opt

(* begin_channel_diagnostic *)
  two_way_recovery_not_supported (*! constructor *),
  tppm_recovery_not_available (*! constructor *),
  functional_unit_not_supported (*! constructor *),
  functional_unit_combination_not_supported (*! constructor *),
  association_reserved (*! constructor *),
  no_reason_given (*! constructor *),
  bcd_absent (*! constructor *) :
    -> tp_begin_channel_diagnostic_Opt
  chan_nat : tp_begin_channel_diagnostic_Opt -> nat
  is_association_reserved : tp_begin_channel_diagnostic_Opt -> bool

(* tp_begin_dialogue_result *)
  accepted (*! constructor *),
  rejected_user (*! constructor *),
  rejected_provider (*! constructor *) :
    -> tp_begin_dialogue_result
  is_accepted, is_rejected, is_rejected_user, is_rejected_provider :
    tp_begin_dialogue_result -> bool

(* tp_assoc_estab_diagnostic_Opt *)
  tp_assoc_estab_diagnostic_Opt (*! constructor *) : bit_string -> tp_assoc_estab_diagnostic_Opt
  TPAEdiag : bool, bool, bool, bool, bool -> tp_assoc_estab_diagnostic_Opt

(*
tp_protocol_version_incompatibility ( 0),    ccr_version_2_not_available ( 1),
contention_winner_assignment_rejected ( 2),  bid_mandatory_rejected ( 3),
no_reason_given ( 4)
 *)
  aed_absent : -> tp_assoc_estab_diagnostic_Opt
  is_absent, pvi_or_ccrv2, cwr_or_bmr, nrg :
    tp_assoc_estab_diagnostic_Opt -> bool

(* bool_Opt *)
  Opt (*! constructor *) : bool -> bool_Opt
  bool_absent (*! constructor *) : -> bool_Opt
  is_absent, get_bool : bool_Opt -> bool

(* bid_result *)
  bid_accepted (*! constructor *),
  bid_rejected (*! constructor *) : -> tp_bid_result
  is_accepted, is_rejected : tp_bid_result -> bool

(* channel_utilisation *)
  one_way_recovery (*! constructor *),
  two_way_recovery (*! constructor *) : -> channel_utilization
  is_two_way : channel_utilization -> bool

(* detach type *)
  free (*! constructor *),
  clean_up (*! constructor *),
  not_used (*! constructor *) : -> detach_type
  is_free, is_clean_up, is_not_used : detach_type -> bool

(* confirmation_urgency *)
  urgent (*! constructor *),
  normal (*! constructor *) : -> confirmation_urgency

(* confirmation_urgency_Opt *)
  Opt (*! constructor *) : confirmation_urgency -> confirmation_urgency_Opt
  cnfu_absent (*! constructor *) : -> confirmation_urgency_Opt
  get_cnfu : confirmation_urgency_Opt -> confirmation_urgency
  is_absent : confirmation_urgency_Opt -> bool

(* contention_winner_assignment *)
  initiator (*! constructor *),
  recipient (*! constructor *) : -> contention_winner_assignment

(* deferred *)
  end_dialogue (*! constructor *),
  grant_control (*! constructor *) : -> tp_defer_type

(* heuristic_report *)
  heuristic_mix (*! constructor *),
  heuristic_hazard (*! constructor *) : -> heuristic_report
  is_heuristic_mix, is_heuristic_hazard : heuristic_report -> bool

(* heuristic_report_Opt *)
  Opt (*! constructor *) : heuristic_report -> heuristic_report_Opt
  hr_absent (*! constructor *) : -> heuristic_report_Opt
  get_hr : heuristic_report_Opt -> heuristic_report
  is_absent : heuristic_report_Opt -> bool

(*  token type  *)
  keep (*! constructor *),
  regular (*! constructor *),
  two_way_recovery (*! constructor *) : -> reason
  is_keep, is_regular, is_two_way_recovery : reason -> bool

(* mappings *)
  abortRI (*! constructor *),
  dataRI (*! constructor *),
  commitRI (*! constructor *),
  commitRC (*! constructor *),
  rollbackRI (*! constructor *),
  rollbackRC (*! constructor *),
  recover_doneRC (*! constructor *),
  recover_commitRI (*! constructor *),
  recover_readyRI (*! constructor *) : -> mapping

(* functional units *)
  functional_units (*! constructor *) : bit_string -> functional_units

(*
   { polarized ( 0), shared ( 1), commit+chained ( 2),
     commit+unchained ( 3), handshake ( 4), recovery ( 5) }
```

```
 *)
  recovery_selected,
  dialogue_selected,
  shared_selected,
  polarized_selected,
  handshake_selected,
  commit_selected,
  unchained_selected,
  chained_selected :
    functional_units -> bool
  recovery : -> functional_units

(* functional_units_Opt *)
  Opt (*! constructor *) : functional_units -> functional_units_Opt
  fu_absent (*! constructor *) : -> functional_units_Opt
  functional_units : functional_units_Opt -> functional_units
  is_absent : functional_units_Opt -> bool

(*  abort_type  *)
  user (*! constructor *),
  provider (*! constructor *) : -> tp_abort_type

eqns
    forall
      bcdO : tp_begin_channel_diagnostic_Opt,
      bdc : tp_begin_dialogue_confirmation,
      bddO : tp_begin_dialogue_diagnostic_Opt,
      bdr : tp_begin_dialogue_result,
      br : tp_bid_result,
      b, pvi, ccrv2, cwr, bmr, n : bool,
      cnfu : confirmation_urgency,
      dt : detach_type,
      tokr : reason,
      hr : heuristic_report,
      fu : functional_units,
      bs : bit_string
  ofsort confirmation_urgency
    get_cnfu(Opt(cnfu)) = cnfu
  ofsort heuristic_report
    get_hr(Opt(hr)) = hr
  ofsort functional_units
    functional_units(Opt(fu)) = fu;
    recovery = functional_units(false + (false + (false + (false + (false + string(true))))));
  ofsort nat
    chan_nat(two_way_recovery_not_supported) = 0;
    chan_nat(tppm_recovery_not_available) = 1;
    chan_nat(functional_unit_not_supported) = 2;
    chan_nat(functional_unit_combination_not_supported) = 3;
    chan_nat(association_reserved) = 4;
    chan_nat(no_reason_given) = 5;
    chan_nat(bcd_absent) = 6;
    dial_nat(recipient_unknown) = 0;
    dial_nat(recipient_tpsu_unknown) = 1;
    dial_nat(tpsu_not_available_permanent) = 2;
    dial_nat(tpsu_not_available_transient) = 3;
    dial_nat(recipient_tpsu_title_required) = 4;
    dial_nat(functional_unit_not_supported) = 5;
    dial_nat(functional_unit_combination_not_supported) = 6;
    dial_nat(association_reserved) = 7;
    dial_nat(no_reason_given) = 8;
    dial_nat(bdd_absent) = 9;
  ofsort bool
    is_begin_trans_rej(begin_transaction_reject) = true;
    is_begin_trans_rej(permanent_failure) = false;
    is_begin_trans_rej(transient_failure) = false;
    is_begin_trans_rej(protocol_error) = false;
    is_begin_trans_rej(end_dialogue_collision) = false;
    is_begin_trans_rej(begin_transaction_end_dialogue_collision) = false;
    is_always(always) = true;
    is_always(negative) = false;
    is_negative(bdc) = not(is_always(bdc));
    is_functional_unit_not_supported(bdd0) = dial_nat(bdd0) eq 5;
    is_association_reserved(bdd0) = dial_nat(bdd0) eq 7;
    is_association_reserved(bcd0) = chan_nat(bcd0) eq 4;
    is_rejected_user(accepted) = false;
    is_rejected_user(rejected_user) = true;
    is_rejected_user(rejected_provider) = false;
    is_rejected_provider(accepted) = false;
    is_rejected_provider(rejected_user) = false;
    is_rejected_provider(rejected_provider) = true;
    is_rejected(bdr) = is_rejected_user(bdr) or is_rejected_provider(bdr);
    is_accepted(bdr) = not(is_rejected(bdr));
    is_rejected(bid_accepted) = false;
    is_rejected(bid_rejected) = true;
    is_accepted(br) = not(is_rejected(br));
    is_two_way(one_way_recovery) = false;
    is_two_way(two_way_recovery) = true;
    is_free(dt) = not(is_clean_up(dt)) and not(is_not_used(dt));
    is_clean_up(free) = false;
    is_clean_up(clean_up) = true;
    is_clean_up(not_used) = false;
    is_not_used(free) = false;
    is_not_used(clean_up) = false;
    is_not_used(not_used) = true;
    is_regular(tokr) = not(is_keep(tokr)) and not(is_two_way_recovery(tokr));
    is_keep(regular) = false;
    is_keep(keep) = true;
    is_keep(two_way_recovery) = false;
    is_two_way_recovery(regular) = false;
    is_two_way_recovery(keep) = false;
    is_two_way_recovery(two_way_recovery) = true;
    is_heuristic_mix(heuristic_mix) = true;
    is_heuristic_mix(heuristic_hazard) = true;
    is_heuristic_hazard(hr) = not(is_heuristic_mix(hr));
    get_bool(Opt(b)) = b;
    is_absent(Opt(b)) = false;
    is_absent(bool_absent) = true;
    is_absent(Opt(cnfu)) = false;
    is_absent(cnfu_absent) = true;
    is_absent(Opt(hr)) = false;
    is_absent(hr_absent) = true;
    is_absent(Opt(fu)) = false;
    is_absent(fu_absent) = true;
(*  definitons for functional units*)
    polarized_selected(functional_units(bs)) = get_bit(0, bs);
    shared_selected(functional_units(bs)) = get_bit(1, bs);
    chained_selected(functional_units(bs)) = get_bit(2, bs);
    unchained_selected(functional_units(bs)) = get_bit(3, bs);
    handshake_selected(functional_units(bs)) = get_bit(4, bs);
    recovery_selected(functional_units(bs)) = get_bit(5, bs);
    dialogue_selected(fu) = polarized_selected(fu) or shared_selected(fu);
    commit_selected(fu) = chained_selected(fu) or unchained_selected(fu);
(* Definitions for Association Establishment diagnostic *)
    pvi_or_ccrv2(tp_assoc_estab_diagnostic_Opt(bs)) =
      get_bit(0, bs) or get_bit(1, bs);
    cwr_or_bmr(tp_assoc_estab_diagnostic_Opt(bs)) =
      get_bit(2, bs) or get_bit(3, bs);
    is_absent(tp_assoc_estab_diagnostic_Opt(bs)) = all_false(bs);
(* introduced *)
    nrg(tp_assoc_estab_diagnostic_Opt(bs)) = get_bit(4,bs);
  ofsort
    tp_assoc_estab_diagnostic_Opt
            (* 8.5.5a - If none apply, omit field *)
    TPAEdiag(pvi, ccrv2, cwr, bmr, n) =
      tp_assoc_estab_diagnostic_Opt(
        pvi + (ccrv2 + (cwr + (bmr + string(n)))));
    aed_absent = tp_assoc_estab_diagnostic_Opt(string(false));
  ofsort tp_begin_dialogue_diagnostic_Opt
    not(is_association_reserved(bdd0)) => suppress_assoc_resv(bdd0) = bdd0;
    is_association_reserved(bdd0) =>
      suppress_assoc_resv(bdd0) = no_reason_given
endtype
(* ----------------------------------------------------------------------
module TP_parameters is BitString

/* P_abort_diagnostic */
    type tp_P_abort_diagnostic is
        permanent_failure
    |   transient_failure
    |   begin_transaction_reject
    |   end_dialogue_collision
    |   begin_transaction_end_dialogue_collision
    |   protocol_error
    endtype

    function is_begin_trans_rej (PAD: tp_P_abort_diagnostic) : bool is
        PAD match begin_transaction_reject
    endfunc

/* begin_dialogue_confirmation */
    type tp_begin_dialogue_confirmation is
        always
    |   negative
    endtype

    function is_always (BDC: tp_begin_dialogue_confirmation) : bool is
        BDC match always
    endfunc

    function is_negative (BDC: tp_begin_dialogue_confirmation) : bool is
        BDC match negative
    endfunc

/* begin_dialogue_diagnostic */
    type tp_begin_dialogue_diagnostic_Opt is
        recipient_unknown
    |   recipient_tpsu_unknown
    |   tpsu_not_available_permanent
    |   tpsu_not_available_transient
    |   recipient_tpsu_title_required
    |   functional_unit_not_supported
    |   functional_unit_combination_not_supported
    |   association_reserved
    |   no_reason_given
    |   bdd_absent
    endtype
/* no translation is needed for dial_nat */
    function dial_nat (BDD: tp_begin_dialogue_diagnostic_Opt) : nat is
        case BDD in
            recipient_unknown -> 0
        |   recipient_tpsu_unknown -> 1
        |   tpsu_not_available_permanent -> 2
        |   tpsu_not_available_transient -> 3
        |   recipient_tpsu_title_required -> 4
        |   functional_unit_not_supported -> 5
        |   functional_unit_combination_not_supported -> 6
        |   association_reserved -> 7
        |   no_reason_given -> 8
        |   bdd_absent -> 9
        endcase
    endfunc

    function is_functional_unit_not_supported (BDD: tp_begin_dialogue_diagnostic_Opt) : bool is
        BDD match functional_unit_not_supported
    endfunc

    function is_association_reserved (BDD: tp_begin_dialogue_diagnostic_Opt) : bool is
        (BDD match association_reserved) or (BDD match recipient_tpsu_title_required)
    endfunc

    function suppress_assoc_resv (BDD: tp_begin_dialogue_diagnostic_Opt) :
                                                    tp_begin_dialogue_diagnostic_Opt is
        if is_association_reserved (BDD) then BDD
        else no_reason_given
        endif
    endfunc
```

```
/* begin_channel_diagnostic */
    type tp_begin_channel_diagnostic_Opt is
            two_way_recovery_not_supported
        |   tppm_recovery_not_available
        |   functional_unit_not_supported
        |   functional_unit_combination_not_supported
        |   association_reserved
        |   no_reason_given
        |   bcd_absent
    endtype
/* not translation is needed for chan_nat */
    function chan_nat (BCD: tp_begin_channel_diagnostic_Opt) : nat is
        case BCD in
            two_way_recovery_not_supported -> 0
        |   tppm_recovery_not_available -> 1
        |   functional_unit_not_supported -> 2
        |   functional_unit_combination_not_supported -> 3
        |   association_reserved -> 4
        |   no_reason_given -> 5
        |   bcd_absent -> 6
        endcase
    endfunc

    function is_association_reserved (BCD: tp_begin_channel_diagnostic_Opt) : bool is
        BCD match  association_reserved
    endfunc

/* tp_begin_dialogue_result */
    type tp_begin_dialogue_result is
            accepted
        |   rejected_user
        |   rejected_provider
    endtype
/* not really need to specify this function */
    function is_accepted (BDR: tp_begin_dialogue_result) : bool is
        BDR match accepted
    endfunc
/* not really need to specify this function */
    function is_rejected (BDR: tp_begin_dialogue_result) : bool is
        (BDR match rejected_user) or (BDR match rejected_provider)
    endfunc
/* not really need to specify this function */
    function is_rejected_user (BDR: tp_begin_dialogue_result) : bool is
        BDR match rejected_user
    endfunc
/* not really need to specify this function */
    function is_rejected_provider (BDR: tp_begin_dialogue_result) : bool is
        BDR match rejected_provider
    endfunc

/* tp_assoc_estab_diagnostic_Opt */
    type tp_assoc_estab_diagnostic_Opt is
        tp_assoc_estab_diagnostic_Opt (BS: bit_string)
    endtype

    function TPAEdiag (pvi, ccrv2, cwr, bmr, nrg: bool) : tp_assoc_estab_diagnostic_Opt is
        tp_assoc_estab_diagnostic_Opt(pvi + (ccrv2 + (cwr + (bmr + string(nrg)))))
    endfunc

    function aed_absent : tp_assoc_estab_diagnostic_Opt is
        tp_assoc_estab_diagnostic_Opt(string(false))
    endfunc

    function is_absent (AEDO: tp_assoc_estab_diagnostic_Opt) : bool is
        all_false (select BS in AEDO)
    endfunc

    function pvi_or_ccrv2 (AEDO: tp_assoc_estab_diagnostic_Opt) : bool is
        get_bit(0, bs) or get_bit(1, bs)
/* let pvi + (ccrv2 + (cwr + (bmr + string(nrg)))) = AEDO in pvi or ccrv2 */
    endfunc

    function cwr_or_bmr (AEDO: tp_assoc_estab_diagnostic_Opt) : bool is
        get_bit(2, bs) or get_bit(3, bs)
/* let pvi + (ccrv2 + (cwr + (bmr + string(nrg)))) = AEDO in cwr or bmr */
    endfunc

/* bool_Opt */
    type bool_Opt is
        Opt (B: bool)
    |   bool_absent
    endtype

    function is_absent (BO: bool_Opt) : bool is BO match bool_absent endfunc

    function get_bool (BO: bool_Opt) : bool is select B in BO endfunc

/* bid_result */
    type tp_bid_result is
        bid_accepted
    |   bid_rejected
    endtype

    function is_accepted (BR: tp_bid_result) : bool is BR match bid_accepted endfunc

    function is_rejected (BR: tp_bid_result) : bool is BR match bid_rejected endfunc

/* channel_utilisation */
    type channel_utilization is
        one_way_recovery
    |   two_way_recovery
    endtype

    function is_two_way (CU: channel_utilization) : bool is CU match two_way_recovery endfunc

/* detach type */
    type detach_type is
```

```
        free
    |   clean_up
    |   not_used
    endtype

    function is_free (DT: detach_type) : bool is DT match free endfunc
    function is_clean_up (DT: detach_type) : bool is DT match clean_up endfunc
    function is_not_used (DT: detach_type) : bool is DT match not_used endfunc

/* confirmation_urgency */
    type confirmation_urgency is
        urgent
    |   normal
    endtype

/* confirmation_urgency_Opt */
    type confirmation_urgency_Opt is
        Opt (CU: confirmation_urgency)
    |   cnfu_absent
    endtype

    function get_cnfu (CUO: confirmation_urgency_Opt) : confirmation_urgency is
        select CU in CUO endfunc

    function is_absent (CUO: confirmation_urgency_Opt) : bool is CUO match cnfu_absent endfunc

/* contention_winner_assignment */
    type contention_winner_assignment is
        initiator
    |   recipient
    endtype

/* deferred */
    type tp_defer_type is
        end_dialogue
    |   grant_control
    endtype

/* heuristic_report */
    type heuristic_report is
        heuristic_mix
    |   heuristic_hazard
    endtype

    function is_heuristic_mix (HR: heuristic_report) is HR match heuristic_mix endfunc
    function is_heuristic_hazard (HR: heuristic_report) is HR match heuristic_hazard endfunc

/* heuristic_report_Opt */
    type  heuristic_report_Opt is
        Opt (HR: heuristic_report)
    |   hr_absent
    endtype

    function get_hr (HRO: heuristic_report_Opt) : heuristic_report is select HR in HRO endfunc
    function is_absent (HRO: heuristic_report_Opt) : bool HRO match hr_absent endfunc

/*  token type  */
    type reason is
        keep
    |   regular
    |   two_way_recovery
    endtype

    function is_keep (R: reason) : bool is R match keep endfunc
    function is_regular (R: reason) : bool is R match regular endfunc
    function is_two_way_recovery (R: reason) : bool is R match two_way_recovery  endfunc

/* mappings */
    type mapping is
        abortRI
    |   dataRI
    |   commitRI
    |   commitRC
    |   rollbackRI
    |   rollbackRC
    |   recover_doneRC
    |   recover_commitRI
    |   recover_readyRI
    endtype

/* functional units */
    type functional_units is
        functional_units (BS: bit_string)
    |   my_functional_units /* shifted from local_values module */
    endtype

    function polarized_selected (FU: functional_units) : bool is get_bit(0, bs) endfunc
    function shared_selected (FU: functional_units) : bool is get_bit(1, bs) endfunc
    function chained_selected  (FU: functional_units) : bool is get_bit(2, bs) endfunc
    function unchained_selected (FU: functional_units) : bool is get_bit(3, bs) endfunc
    function handshake_selected (FU: functional_units) : bool is get_bit(4, bs) endfunc
    function recovery_selected (FU: functional_units) : bool is get_bit(5, bs) endfunc
    function dialogue_selected (FU: functional_units) : bool is get_bit(0, bs) or get_bit(1, bs) endfunc
    function commit_selected (FU: functional_units) : bool is get_bit(2, bs) or get_bit(3, bs) endfunc

    function recovery : functional_units is
        functional_units(false + (false + (false + (false + (false + string(true))))))
    endfunc

/* functional_units_Opt */
    type functional_units_Opt is
        Opt (FU: functional_units)
    |   fu_absent
    endfunc

    function functional_units (FUO: functional_units_Opt) : functional_units is select FU in FUO endfunc
    function is_absent (FUO: functional_units_Opt) : bool is FUO match fu_absent endfunc
```

```
/*  abort_type  */
    type abort_type is
        user
    |   provider
    endtype

endmod
========================================================================== *)


type parameters is
  TP_parameters,
  CCR_parameters,
  ACSE_parameters,
  SAF_parameters,
  presentation_parameters,
  PDU_Queues
(*  All user-data appears as sequence of PDUs i.e. 'PDUqueue's  *)
sorts
  correlator_Opt,
  quality_of_service_Opt,
  TPSU_title_Opt,
  recovery_context_handle,
  recovery_context_handle_Opt
opns
(* correlator_Opt *)
  correlator (*! constructor *) : nat -> correlator_Opt
  nat : correlator_Opt -> nat
  dcc_absent (*! constructor *) : -> correlator_Opt
  is_absent : correlator_Opt -> bool
  inc_dcc : correlator_Opt -> correlator_Opt
  _eq_, _ne_ : correlator_Opt, correlator_Opt -> bool

(* TPSU_title_Opt *)
  TPSU_title (*! constructor *) : Integer -> TPSU_title_Opt
  TPSU_absent (*! constructor *) : -> TPSU_title_Opt

(* Quality of Service Definitions *)
  quality_of_service (*! constructor *) : QOS_IS08326 -> quality_of_service_Opt
 (* QOS used in TP, ACSE and Presentation are all the session qos *)
  qos_absent (*! constructor *) : -> quality_of_service_Opt
  is_absent : quality_of_service_Opt -> bool
  get_qos : quality_of_service_Opt -> QOS_IS08326

(* recovery_context_handle *)
  recovery_context_handle (*! constructor *) : OctetString -> recovery_context_handle
  _eq_ : recovery_context_handle, recovery_context_handle_Opt -> bool

(* recovery_context_handle_Opt *)
  Opt (*! constructor *) : recovery_context_handle -> recovery_context_handle_Opt
  rch_absent (*! constructor *) : -> recovery_context_handle_Opt
  is_absent : recovery_context_handle_Opt -> bool

(*user-data*)
```

```
  ud_absent : -> PDUqueue

eqns
    forall
      q_o_s : QOS_ISO8326,
      rch : recovery_context_handle,
      os1, os2 : OctetString,
      n1, n2 : nat,
      dcc1, dcc2 : correlator_Opt
  ofsort QOS_ISO8326 get_qos(quality_of_service(q_o_s)) = q_o_s;
  ofsort bool
    is_absent(dcc_absent) = true;
    is_absent(correlator(n1)) = false;
    is_absent(qos_absent) = true;
    is_absent(quality_of_service(q_o_s)) = false;
    is_absent(rch_absent) = true;
    is_absent(Opt(rch)) = false;
    rch eq rch_absent = false;
    recovery_context_handle(os1) eq Opt(recovery_context_handle(os2)) = os1 eq os2;
    correlator(n1) eq correlator(n2) = n1 eq n2;
    dcc1 eq dcc_absent = true;
    dcc_absent eq correlator(n1) = false;
    dcc1 ne dcc2 = not(dcc1 eq dcc2);
  ofsort nat nat(correlator(n1)) = n1;
  ofsort correlator_Opt inc_dcc(correlator(n1)) = correlator(succ(n1));
  ofsort PDUqueue (*user-data*) ud_absent = emptyPDU;
endtype
(* ---------------------------------------------------------------------------
module parameters is
  TP_parameters,
  CCR_parameters,
  ACSE_parameters,
  SAF_parameters,
  presentation_parameters,
  PDU_Queues

/* correlator */
    type correlator_Opt is
        correlator (N: nat)
    |   dcc_absent
    endtype

    function nat (CO: correlator_Opt) : nat is select N in CO endfunc

    function is_absent (CO: correlator_Opt) : bool is CO match dcc_absent endfunc

    function inc_dcc (CO: correlator_Opt) : correlator_Opt is correlator ((select N in CO) + 1)endfunc
/* _eq_, such are defined, is not syntactical equality */
    function _eq_ (CO1, CO2: correlator_Opt) : bool is
        case CO1 in
           correlator(N:=n1: nat) -> (CO2 match dcc_absent) orelse (n1 eq (select N in CO2))
        |  dcc_absent -> (CO2 match dcc_absent)
        endcase
    endfunc
/* TPSU_title_Opt */
    type TPSU_title_Opt is
        TPSU_title (I: Integer)
    |   TPSU_absent
    endtype

/* Quality of Service Definitions */
    type quality_of_service_Opt is
        quality_of_service (QOS: QOS_ISO8326)
 /* QOS used in TP, ACSE and Presentation are all the session qos */
    |   qos_absent
    endtype

    function is_absent (Q: quality_of_service_Opt) : bool is Q match qos_absent endfunc
    function get_qos(Q: quality_of_service_Opt) : QOS_ISO8326 is select QOS in Q endfunc

/* recovery_context_handle */
    type recovery_context_handle is
         recovery_context_handle (OS: OctetString)
    endtype

    function _eq_ (RCH: recovery_context_handle, RCHO: recovery_context_handle_Opt) : bool is
        (select OS in RCH) eq (select OS in (select RCH in RCHO))
    endfunc

/* recovery_context_handle_Opt */
    type recovery_context_handle_Opt is
        Opt (RCH: recovery_context_handle)
    |   rch_absent
    endtype

    function is_absent (RCHO: recovery_context_handle_Opt) : bool is RCHO match rch_absent endfunc

/* user-data */
    function ud_absent : PDUqueue is emptyPDU endfunc
endmod
========================================================================== *)


type CCR_parameters is addressing
sorts
  atomic_action_identifier,
  atomic_action_identifier_Opt,
  branch_identifier,
  branch_identifier_Opt,
  suffix,
  recovery_state,       (* values used on request and indication  *)
  recovery_response     (* values used on response and confirm  *)
opns

(* atomic_action_identifier *)
  atomic_action_identifier (*! constructor *) : AE_title, suffix -> atomic_action_identifier
  masters_name : atomic_action_identifier -> AE_title
  suffix : atomic_action_identifier -> suffix
  _eq_ : atomic_action_identifier, atomic_action_identifier -> bool

(* atomic_action_identifier_Opt *)
  Opt (*! constructor *) : atomic_action_identifier -> atomic_action_identifier_Opt
  aaid_absent (*! constructor *) : -> atomic_action_identifier_Opt

(* branch_identifier *)
  branch_identifier (*! constructor *) : AE_title, suffix -> branch_identifier
  superiors_name : branch_identifier -> AE_title
  suffix : branch_identifier -> suffix
  _eq_ : branch_identifier, branch_identifier -> bool

(* branch_identifier_Opt *)
  Opt (*! constructor *) : branch_identifier -> branch_identifier_Opt
  brid_absent (*! constructor *) : -> branch_identifier_Opt

(* Is specified by ACSE-1.AE-title - ref ISO 8650/X.227 *)

(* suffix *)
  suffix (*! constructor *) : OctetString -> suffix
  suffix (*! constructor *) : Integer ->  suffix
  _eq_ : suffix, suffix -> bool

(* recovery-state *)
  commit (*! constructor *),
  ready (*! constructor *) : -> recovery_state
  is_commit, is_ready : recovery_state -> bool

(* recovery_response *)
  done (*! constructor *),
  unknown (*! constructor *),
  retry_later (*! constructor *) : -> recovery_response
  is_done, is_unknown, is_retry_later : recovery_response -> bool

eqns
    forall
      aet, aet1, aet2 : AE_title,
      aaid : atomic_action_identifier,
      brid : branch_identifier,
      sfx, sfx1, sfx2 : suffix,
      s1, s2 : OctetString,
      i1, i2 : Integer,
      rs : recovery_state,
      rr : recovery_response
  ofsort AE_title
    masters_name(atomic_action_identifier(aet, sfx)) = aet;
    superiors_name(branch_identifier(aet, sfx)) = aet;
  ofsort suffix
    suffix(atomic_action_identifier(aet, sfx)) = sfx;
    suffix(branch_identifier(aet, sfx)) = sfx;
  ofsort bool
    is_commit(commit) = true;
    is_commit(ready) = false;
    is_ready(rs) = not(is_commit(rs));
    is_done(done) = true;
    is_done(unknown) = false;
    is_done(retry_later) = false;
    is_unknown(done) = false;
    is_unknown(unknown) = true;
    is_unknown(retry_later) = false;
    is_retry_later(rr) = not(is_done(rr)) and not(is_unknown(rr));
(* added *)
    aet1 eq aet2, sfx1 eq sfx2 =>
        atomic_action_identifier(aet1, sfx1) eq atomic_action_identifier(aet2, sfx2) = true;
    atomic_action_identifier(aet1, sfx1) eq atomic_action_identifier(aet2, sfx2) = false;
    aet1 eq aet2, sfx1 eq sfx2 =>
        branch_identifier(aet1, sfx1) eq branch_identifier(aet2, sfx2) = true;
    branch_identifier(aet1, sfx1) eq branch_identifier(aet2, sfx2) = false;
    s1 eq s2 => suffix(s1) eq suffix(s2) = true;
    i1 eq i2 => suffix(i1) eq suffix(i2) = true;
    sfx1 eq sfx2 = false;
endtype
(* ------------------------------------------------------------------------

module CCR_parameters is addressing

    type atomic_action_identifier is
        atomic_action_identifier (AET: AE_title, S: suffix)
    endtype

    function masters_name (AAI: atomic_action_identifier) : AE_title is select AET in AAI endfunc
    function suffix (AAI: atomic_action_identifier) : suffix is selectS in AAI endfunc

    type atomic_action_identifier_Opt is
        Opt (AAI: atomic_action_identifier)
    |   aaid_absent
    endtype

    type branch_identifier is
        branch_identifier (AET: AE_title, S: suffix)
    endtype

    function superiors_name (BI: branch_identifier) : AE_title is select AET in BI endfunc
    function suffix (BI: branch_identifier) : suffix is select S in BI endfunc

    type branch_identifier_Opt is
        Opt (BI: branch_identifier)
    |   brid_absent
    endtype

/* syntactical equality will be provided */
    type suffix is
        suffix (OS: OctetString)
    |   suffix (I: Integer)
    endtype

    type recovery_state is
```

```
        commit
    |   ready
    endtype

    function is_commit (RS: recovery_state) : bool is RS match commit endfunc
    function is_ready(RS : recovery_state) : bool is RS match ready endfunc

    type recovery_response is
        done
    |   unknown
    |   retry_later
    endtype

    function is_done (RR: recovery_reponse) : bool is RR match done endfunc
    function is_unknown (RR: recovery_reponse) : bool is RR match unknown endfunc
    function is_retry_later (RR: recovery_reponse) : bool is RR match retry_later endfunc
endmod
========================================================================== *)


(* Supporting datatypes for acse service definition *)
(*
   The acse sps are tailored to the requirements specified for use
   with tp
 *)
type addressing is external_datatypes
sorts
  AP_title_Opt,
  AE_qual_Opt,
  API_id_Opt,   (*  application process invocation identifier  *)
  AEI_id_Opt,   (*  application entity invocation identifier  *)
  address,      (*  the addressing parameters on AAssociate, TPBeginDialogue, etc.  *)
  AE_title,     (*  application entity title  *)
  AE_title_Opt
opns
(* AP_title_Opt *)
  Opt (*! constructor *) : AP_title -> AP_title_Opt
  apt_absent (*! constructor *) : -> AP_title_Opt
  _<=_ : AP_title_Opt, AP_title_Opt -> bool

(* AE_qual_Opt *)
  Opt (*! constructor *) : AE_qual -> AE_qual_Opt
  aeq_absent (*! constructor *) : -> AE_qual_Opt
  _<=_ : AE_qual_Opt, AE_qual_Opt -> bool

(* API_id_Opt *)
  API_id (*! constructor *) : Integer -> API_id_Opt
  apii_absent (*! constructor *) : -> API_id_Opt
  _eq_, _<=_ : API_id_Opt, API_id_Opt -> bool

(* AEI_id_Opt *)
  AEI_id (*! constructor *) : Integer -> AEI_id_Opt
  aeii_absent (*! constructor *) : -> AEI_id_Opt
  _eq_, _<=_ : AEI_id_Opt, AEI_id_Opt -> bool
```

```
(* AE_title *)
(* As defined in ISO 7498-3 *)
  AE_title (*! constructor *) : AP_title, AE_qual -> AE_title
  get_apt : AE_title -> AP_title
  get_aeq : AE_title -> AE_qual
  _eq_ : AE_title, AE_title -> bool

(* AE_title_Opt *)
  Opt (*! constructor *) : AE_title -> AE_title_Opt
  aet_absent (*! constructor *) : -> AE_title_Opt

(* address *)
  address (*! constructor *) : AP_title_Opt, API_id_Opt, AE_qual_Opt, AEI_id_Opt -> address
  get_apt : address -> AP_title_Opt
  get_aeq : address -> AE_qual_Opt
  get_apii : address -> API_id_Opt
  get_aeii : address -> AEI_id_Opt
(* <= true if RHS is more specific version of LHS *)
  _<=_ : address, address -> bool
  has_aet : address -> bool
  get_aetO : address -> AE_title_Opt

eqns
    forall
      apt, apt1 : AP_title,
      aptO, aptO1 : AP_title_Opt,
      aeq, aeq1 : AE_qual,
      aeqO, aeqO1 : AE_qual_Opt,
      apiiO, apiiO1 : API_id_Opt,
      aeiiO, aeiiO1 : AEI_id_Opt,
      aet, aet1 : AE_title,
      aetO, aetO1 : AE_title_Opt,
      addr : address,
      n, m : Integer
  ofsort bool
    apt_absent <= aptO = true;
    Opt(apt) <= Opt(apt1) = apt eq apt1;
    Opt(apt) <= apt_absent = false;
    aeq_absent <= aeqO = true;
    Opt(aeq) <= Opt(aeq1) = aeq eq aeq1;
    Opt(aeq) <= aeq_absent = false;
    API_id(n) eq API_id(m) = n eq m;
    API_id(n) eq apii_absent = false;
    apii_absent eq API_id(m) = false;
    apii_absent eq apii_absent = true;
    apiiO <= apiiO1 = (apiiO eq apii_absent) or (apiiO eq apiiO1);
    AEI_id(n) eq AEI_id(m) = n eq m;
    AEI_id(n) eq aeii_absent = false;
    aeii_absent eq AEI_id(m) = false;
    aeii_absent eq aeii_absent = true;
    aeiiO <= aeiiO1 = (aeiiO eq aeii_absent) or (aeiiO eq aeiiO1);
    AE_title(apt, aeq) eq AE_title(apt1, aeq1) =
      (apt eq apt1) and (aeq eq aeq1);
    address(apt0, apii0, aeq0, aeii0) <= address(apt01, apii01, aeq01, aeii01) =
      (apt0 <= apt01)
      and
      (apii0 <= apii01)
      and
      (aeq0 <= aeq01)
      and
      (aeii0 <= aeii01);
    has_aet(address(apt0, apii0, aeq0, aeii0)) =
      not(apt0 <= apt_absent) and not(aeq0 <= aeq_absent);
(* selection operators for address block*)
  ofsort AP_title_Opt get_apt(address(apt0, apii0, aeq0, aeii0)) = apt0;
  ofsort AE_qual_Opt get_aeq(address(apt0, apii0, aeq0, aeii0)) = aeq0;
  ofsort API_id_Opt get_apii(address(apt0, apii0, aeq0, aeii0)) = apii0;
  ofsort AEI_id_Opt get_aeii(address(apt0, apii0, aeq0, aeii0)) = aeii0;
  ofsort AP_title get_apt(AE_title(apt, aeq)) = apt;
  ofsort AE_qual get_aeq(AE_title(apt, aeq)) = aeq;
  ofsort AE_title_Opt
    get_aet0(address(apt_absent, apii0, aeq0, aeii0)) = aet_absent;
    get_aet0(address(apt0, apii0, aeq_absent, aeii0)) = aet_absent;
    get_aet0(address(Opt(apt), apii0, Opt(aeq), aeii0)) =
      Opt(AE_title(apt, aeq));
endtype
(* ---------------------------------------------------------------------------

module addressing is external_datatypes

    type AP_title_Opt is
        Opt (APT: AP_title)
    |   apt_absent
    endtype

    function _<=_ (APTO1: AP_title_Opt, APTO2: AP_title_Opt) : bool is
        (APTO1 eq apt_absent) or (APTO1 eq APTO2)
    endfunc

    type AE_qual_Opt is
        Opt (AEQ: AE_qual)
    |   aeq_absent
    endtype

    function _<=_ (AEQO1: AE_qual_Opt, AEQO2: AE_qual_Opt) : bool is
        (AEQO1 eq aeq_absent) or (AEQO1 eq AEQO2)
    endfunc

    type API_id_Opt is
        API_id (I: Integer)
    |   apii_absent
    endtype

/* syntactical equality is automatically generated */
    function _<=_ (APIIO1: API_id_Opt, APIIO2: API_id_Opt) : bool is
        (APIIO1 eq apii_absent) or (APIIO1 eq APIIO2)
    endfunc
    type AEI_id_Opt is
        AEI_id (I: Integer)
    |   aeii_absent
    endtype

/* syntactical equality will be automatically generated */
    function _<=_ (AEII01: AEI_id_Opt, AEII02: AEI_id_Opt) : bool is
        (AEII01 eq aeii_absent) or (AEII01 eq AEII02)
    endfunc

    type AE_title is
        AE_title (APT: AP_title, AEQ: AE_qual)
    |   my_aet /* shifted here from local_values type */
    endtype

    function get_apt (AET: AE_title) : AP_title is select APT in AET endfunc
    function get_aeq (AET: AE_title) : AE_qual is select AEQ in AET endfunc
/* syntactical equality will be automatically generated */

    type AE_title_Opt is
        Opt (AET: AE_title)
    |   aet_absent
    endtype

    type address is
        address (APTO: AP_title_Opt, APIIO: API_id_Opt, AEQO: AE_qual_Opt, AEIIO: AEI_id_Opt)
    |   my_addr /* shifted here from local_values type */
    endtype

    function get_apt (A: address) : AP_title_Opt is select APTO in A endfunc
    function get_aeq (A: address) : AE_qual_Opt is select AEQ in A endfunc
    function get_apii (A: address) : API_id_Opt is select APIIO in A endfunc
    function get_aeii (A: address) : AEI_id_Opt is select AEIIO in A endfunc

    function _<=_ (A1: address, A2: address) : bool is
        ((select APTO in A1) <= (select APTO in A2)) and
        ((select APIIO in A1) <= (select APIIO in A2)) and
        ((select AEQO in A1) <= (select AEQO in A2)) and
        ((select AEIIO in A1) <= (select AEIIO in A2))
    endfunc

    function has_aet (A: address) : bool is
        not((select APTO in A) <= apt_absent) and not((select AEQO in A) <= aeq_absent)
/*      not((select APTO in A) match apt_absent) and not((select AEQO in A) match aeq_absent) */
    endfunc

    function get_aetO (A: address) : AE_title_Opt is
        if ((select APTO in A) match apt_absent) or
           ((select APIIO in A) match aeq_absent) then  aet_absent
        else Opt(AE_title(select APT in (select APTO in A), select AEQ in (select APIIO in A))
        endif
    endfunc
endmod
========================================================================= *)

type ACSE_parameters is addressing, presentation_parameters
sorts
  protocol_version,
  application_context_name,
  a_associate_mode,
  a_associate_result,
  a_associate_result_source,
  a_associate_diagnostic_Opt,
  service_user_diagnostic,
  service_provider_diagnostic,
  a_release_result,
  a_abort_source
opns

(* a_associate_mode *)
  normal (*! constructor *) : -> a_associate_mode
  x410_1984 (*! constructor *) : -> a_associate_mode

(* protocol_version *)
  protocol_version (*! constructor *) : bit_string -> protocol_version
    (*  bit0 represents version1  *)
    (*  <= on protocol versions: the lhs is included in the rhs  *)
  _<=_ : protocol_version, protocol_version -> bool

(* application_context_name *)
  application_context_name (*! constructor *) : bit_string -> application_context_name
    (*  bit0 set indicates that CCR is in the application context  *)
    (*  acn1 <= acn2  if acn2 contains acn1  *)
  _<=_ : application_context_name, application_context_name -> bool
  HasCCR : application_context_name -> bool

(* a_associate_result  *)
  accepted (*! constructor *),
  rejected_permanent (*! constructor *),
  rejected_transient (*! constructor *)  : -> a_associate_result
  is_accepted, is_rejected, is_rejected_perm, is_rejected_trans : a_associate_result -> bool

(* a_associate_result_source *)
  ACSE_service_user (*! constructor *),
  ACSE_service_provider (*! constructor *),
  presentation_service_provider (*! constructor *) : -> a_associate_result_source
(* added *)
  is_user, is_provider : a_associate_result_source -> bool

(* service_user_diagnostic *)
  no_reason_given (*! constructor *),
  application_context_name_not_supported (*! constructor *),
  calling_AP_title_unknown (*! constructor *),
  calling_AE_qual_unknown (*! constructor *),
  calling_API_id_unknown (*! constructor *),
  calling_AEI_id_unknown (*! constructor *),
  called_AP_title_unknown (*! constructor *),
  called_AE_qual_unknown (*! constructor *),
  called_API_id_unknown (*! constructor *),
  called_AEI_id_unknown (*! constructor *) : -> service_user_diagnostic

(* service_provider_diagnostic  *)
  no_reason_given (*! constructor *),
  no_common_ACSE_version (*! constructor *) : -> service_provider_diagnostic

(* a_associate_diagnostic_Opt *)
  su_aad_Opt (*! constructor *) : service_user_diagnostic -> a_associate_diagnostic_Opt
  sp_aad_Opt (*! constructor *) : service_provider_diagnostic -> a_associate_diagnostic_Opt
  aad_absent (*! constructor *) : -> a_associate_diagnostic_Opt
  is_absent, is_su_diag, is_sp_diag,
  is_called_addr_not_recognised : a_associate_diagnostic_Opt -> bool

(*  a_release_result *)
  affirmative (*! constructor *),
  negative (*! constructor *) : -> a_release_result
  is_affirmative, is_negative : a_release_result -> bool

(* a_abort_source *)
  ACSE_service_user (*! constructor *),
  ACSE_service_provider (*! constructor *) : -> a_abort_source
  is_user, is_provider : a_abort_source -> bool
eqns
    forall
      aar : a_associate_result,
      arr : a_release_result,
      aas : a_abort_source,
      aars: a_associate_result_source,
      asu : service_user_diagnostic,
      asp : service_provider_diagnostic,
      bs1, bs2 : bit_string
  ofsort bool
    (* protocol_version  - _<=_ true if lhs "contained in" rhs  *)
    protocol_version(bs1) <= protocol_version(bs2) = bs1 implies bs2;
    (* application_context - _<=_ true if lhs "contained in" rhs  *)
    application_context_name(bs1) <= application_context_name(bs2) = bs1 implies bs2;
    HasCCR(application_context_name(bs1)) = get_bit(0, bs1);
    is_rejected_perm(accepted) = false;
    is_rejected_perm(rejected_permanent) = true;
    is_rejected_perm(rejected_transient) = false;
    is_rejected_trans(accepted) = false;
    is_rejected_trans(rejected_permanent) = false;
    is_rejected_trans(rejected_transient) = true;
    is_rejected(aar) = is_rejected_perm(aar) or is_rejected_trans(aar);
    is_accepted(aar) = not(is_rejected(aar));
    is_negative(affirmative) = false;
    is_negative(negative) = true;
    is_affirmative(arr) = not(is_negative(arr));
    is_user(ACSE_service_user of a_associate_result_source) = true;
    is_user(ACSE_service_provider of a_associate_result_source) = false;
    is_user(ACSE_service_user of a_abort_source) = true;
    is_user(ACSE_service_provider of a_abort_source) = false;
    is_provider(aas) = not(is_user(aas));
    is_provider(aars) = not(is_user(aars));
(* diagnostic *)
    is_su_diag(su_aad_Opt(asu)) = true;
    is_su_diag(sp_aad_Opt(asp)) = false;
    is_su_diag(aad_absent) = false;
    is_sp_diag(su_aad_Opt(asu)) = false;
    is_sp_diag(sp_aad_Opt(asp)) = true;
    is_sp_diag(aad_absent) = false;
    is_absent(su_aad_Opt(asu)) = false;
    is_absent(sp_aad_Opt(asp)) = false;
    is_absent(aad_absent) = true;
    is_called_addr_not_recognised(su_aad_Opt(no_reason_given)) = false;
    is_called_addr_not_recognised(
      su_aad_Opt(application_context_name_not_supported)) =
      false;
    is_called_addr_not_recognised(su_aad_Opt(calling_AP_title_unknown)) = false;
    is_called_addr_not_recognised(su_aad_Opt(calling_AE_qual_unknown)) = false;
    is_called_addr_not_recognised(su_aad_Opt(calling_API_id_unknown)) = false;
    is_called_addr_not_recognised(su_aad_Opt(calling_AEI_id_unknown)) = false;
    is_called_addr_not_recognised(su_aad_Opt(called_AP_title_unknown)) = true;
    is_called_addr_not_recognised(su_aad_Opt(called_AE_qual_unknown)) = true;
    is_called_addr_not_recognised(su_aad_Opt(called_API_id_unknown)) = true;
    is_called_addr_not_recognised(su_aad_Opt(called_AEI_id_unknown)) = true;
    is_called_addr_not_recognised(sp_aad_Opt(asp)) = false;
    is_called_addr_not_recognised(aad_absent) = false;
endtype
(* --------------------------------------------------------------------------
module ACSE_parameters is addressing, presentation_parameters

/* a_associate_mode */
    type a_associate_mode is
        normal
    |   x410_1984
    endtype

/* protocol_version */
    type protocol_version is
        protocol_version (BS: bit_string)
    endtype

    function _<=_ (V1: protocol_version, V2: protocol_version) : bool is
        (select BS in V1) implies (select BS in V2)
    endfunc

/* application_context_name */
    type application_context_name is
        application_context_name (BS: bit_string)
    endtype

    function _<=_ (CN1, CN2: application_context_name) : bool is
        (select BS in CN1) implies (select BS in CN2)
    endfunc
    function HasCCR (CN: application_context_name) : bool is
        get_bit(0, select BS in CN)
    endfunc

/* result */
    type result is
        accepted
    |   rejected_permanent
    |   rejected_transient
    endtype

    function is_accepted (R: a_associate_result) : bool is R match accepted endfunc
    function is_rejected (R: a_associate_result) : bool is not(R match accepted) endfunc
    function is_rejected_perm (R: a_associate_result) : bool is R match rejected_permanent endfunc
    function is_rejected_trans (R: a_associate_result) : bool is R match rejected_transient endfunc

/* a_associate_result_source */
    type a_associate_result_source is
        ACSE_service_user
    |   ACSE_service_provider
    |   presentation_service_provider
    endtype

/* service_user_diagnostic */
    type service_user_diagnostic is
        no_reason_given
    |   application_context_name_not_supported
    |   calling_AP_title_unknown
    |   calling_AE_qual_unknown
    |   calling_API_id_unknown
    |   calling_AEI_id_unknown
    |   called_AP_title_unknown
    |   called_AE_qual_unknown
    |   called_API_id_unknown
    |   called_AEI_id_unknown
    endtype

/* service_provider_diagnostic  */
    type service_provider_diagnostic is
        no_reason_given
    |   no_common_ACSE_version
    endtype

/* a_associate_diagnostic_Opt */
    type a_associate_diagnostic_Opt is
        su_aad_Opt (SUD: service_user_diagnostic)
    |   sp_aad_Opt (SPD: service_provider_diagnostic)
    |   aad_absent
    endtype

    function is_absent (AADO: a_associate_diagnostic_Opt) : bool is
        AADO match aad_absent endfunc
    function is_su_diag (AADO: a_associate_diagnostic_Opt) : bool is
        AADO match su_aad_Opt (SUD: service_user_diagnostic) endfunc
    function is_sp_diag (AADO: a_associate_diagnostic_Opt) : bool is
        AADO match su_aad_Opt (SPD: service_provider_diagnostic) endfunc
    function is_called_addr_not_recognised (AADO: a_associate_diagnostic_Opt) : bool is
        case AADO in
            aad_absent
        |   sp_aad_Opt (SPD: service_provider_diagnostic)
        |   su_aad_Opt(no_reason_given)
        |   su_aad_Opt(application_context_name_not_supported)
        |   su_aad_Opt(calling_AP_title_unknown)
        |   su_aad_Opt(calling_AE_qual_unknown)
        |   su_aad_Opt(calling_API_id_unknown)
        |   su_aad_Opt(calling_AEI_id_unknown) -> false
        |   any a_associate_diagnostic_Opt - true
        endcase
    endfunc

/*  a_release_result */
    type a_release_result is
        affirmative
    |   negative
    endtype

    function is_affirmative (RS: a_release_result) : bool is RS match affirmative endfunc
    function is_negative (RS: a_release_result) : bool is RS match negative endfunc

/* a_abort_source */
    type a_abort_source is
        ACSE_service_user
    |   ACSE_service_provider
    endfunc

    function is_user (AS: a_abort_source) : bool is AS match ACSE_service_user endfunc
    function is_provider (AS: a_abort_source) : bool is AS match ACSE_service_provider endfunc
endmod
========================================================================== *)

type SAF_parameters is boolean
sorts status
opns
  free (*! constructor *),
  rollback_indication_expected (*! constructor *),
  rollback_confirm_expected (*! constructor *),
  begin_indication_expected (*! constructor *),
  begin_fear (*! constructor *) : -> status
  IsBeginFear : status -> bool
eqns
  ofsort bool
    IsBeginFear(free) = false;
    IsBeginFear(rollback_indication_expected) = false;
    IsBeginFear(rollback_confirm_expected) = false;
    IsBeginFear(begin_indication_expected) = false;
    IsBeginFear(begin_fear) = true;
endtype
(* --------------------------------------------------------------------------
module  SAF_parameters is Boolean
    type statu is
        free
    |   rollback_indication_expected
    |   rollback_confirm_expected
    |   begin_indication_expected
    |   begin_fear
    endtype

    function IsBeginFear (S: status) : bool is S match begin_fear endfunc
endmod
========================================================================= *)


type presentation_parameters is BitString
sorts
  presentation_context_definition_list,
  session_requirements,
  initial_assignment_of_tokens
opns
(*      Presentation Context Definition List *)
  presentation_context_definition_list (*! constructor *) :
        bit_string -> presentation_context_definition_list
(*
as for application_context_name the only feature of interest to TP is
whether CCR is in the list.  bit0 represents CCR
 *)
  HasCCR : presentation_context_definition_list -> bool

(* session_requirements - schematic form *)
  session_requirements (*! constructor *) : bit_string -> session_requirements
 (*  bit0 is Kernel, bit1 is FullDuplex, bit2 represents CCR requirements  *)
  HasCCRreqs : session_requirements -> bool

(* initial assignment of tokens - only specifiy for minor token *)
  requestor_side (*! constructor *),
  acceptor_side (*! constructor *),
  acceptor_chooses (*! constructor *) : -> initial_assignment_of_tokens
  is_requestor_side, is_acceptor_side, is_acceptor_chooses :
    initial_assignment_of_tokens -> bool

eqns
    forall iaot : initial_assignment_of_tokens, bs : bit_string
  ofsort bool
    HasCCR(presentation_context_definition_list(bs)) = get_bit(0, bs);
    HasCCRreqs(session_requirements(bs)) = get_bit(2, bs);
    is_requestor_side(requestor_side) = true;
    is_requestor_side(acceptor_side) = false;
    is_requestor_side(acceptor_chooses) = false;
    is_acceptor_side(requestor_side) = false;
    is_acceptor_side(acceptor_side) = true;
    is_acceptor_side(acceptor_chooses) = false;
    is_acceptor_chooses(iaot) =
      not(is_requestor_side(iaot)) and not(is_acceptor_side(iaot));
```

```
endtype
(* ---------------------------------------------------------------------------
module presentation_parameters is BitString

/* presentation_context_definition_list */
    type presentation_context_definition_list is
        presentation_context_definition_list (BS: bit_string)
    endtype

    function HasCCR (PCDL: presentation_context_definition_list) : bool is
        get_bit(0, select BS in PCDL) endfunc

/* session_requirements */
    type session_requirements is
        session_requirements (BS: bit_string)
    endtype

    function HasCCRreqs (SR: session_requirements) : bool is
        get_bit(2, select BS in SR) endfunc

/* initial_assignment_of_tokens */
    type initial_assignment_of_tokens is
        requestor_side
    |   acceptor_side
    |   acceptor_chooses
    endtype

    function is_requestor_side (IAT: initial_assignment_of_tokens) : bool is
        IAT match requestor_side endfunc
    function is_acceptor_side (IAT: initial_assignment_of_tokens) : bool is
        IAT match acceptor_side endfunc
    function is_acceptor_chooses (IAT: initial_assignment_of_tokens) : bool is
        IAT match acceptor_chooses endfunc
endmod
========================================================================== *)

(*
 *      H.5.3   PDU
 *)
type BasicPDU is Boolean
sorts PDU, PDUkey
opns
  key : PDU -> PDUkey
  succ (*! constructor *) : PDUkey -> PDUkey
  _eq_, _ne_, _le_ : PDUkey, PDUkey -> bool
  _eq_ : PDU, PDUkey -> bool
endtype
(* ---------------------------------------------------------------------------
module BasicPDU is Boolean
    type PDU is
    endtype

    type PDUkey is
        first_TPpdu /* defined in type TP_PDU */
    |   succ (K: PDUkey)
    endtype
/* no translation is needed for _eq_ */
    function _ne_ (P1: PDUkey, P2: PDUkey) : bool is not (P1 eq P2) endfunc

    function _le_ (P1: PDUkey, P2: PDUkey) : bool
        (P1 match first_TPpdu) or not(P2 match first_TPpdu) or
        ((select K in P1) le (select K in P2))
    endfunc

/* function key is specified in TP_PDU module */
/* function key (P: PDU) : PDUkey is endfunc */

    function _eq_ (P: PDU, K: PDUkey) is key(P) eq K endfunc
endmod
module PDU_equations is PDU
endmod
============================================================================ *)

type PDU_equations is PDU
eqns
    forall pdu : PDU, x, y : PDUkey
  ofsort bool
    first_TPpdu eq first_TPpdu = true;
    succ(x) eq first_TPpdu = false;
    first_TPpdu eq succ(y) = false;
    succ(x) eq succ(y) = x eq y;
    x ne y = not(x eq y);
    first_TPpdu le x = true;
    succ(x) le first_TPpdu = false;
    succ(x) le succ(y) = x le y;
    pdu eq x = key(pdu) eq x;
endtype

type TP_PDU is parameters, BasicPDU
opns
  tp_begin_dialogue_ri (*! constructor *) : (*initiating*)
    TPSU_title_Opt,
    TPSU_title_Opt, (*recipient*)
    functional_units,
    bool_Opt (*begin_transaction_Opt*),
    tp_begin_dialogue_confirmation,
    nat (*correlator*),
    correlator_Opt (*last_partner_identifier*),
    PDUqueue (*user_data*)  ->
      PDU
  tp_begin_dialogue_ri (*! constructor *) :
    functional_units,
    nat (*correlator*),
    channel_utilization,
    correlator_Opt (*last_partner_identifier*)      ->
      PDU
  tp_begin_dialogue_rc (*! constructor *) :
    functional_units_Opt,
    tp_begin_dialogue_result,
    tp_begin_dialogue_diagnostic_Opt,
    nat (*correlator*),
    PDUqueue (*user_data*)  ->
      PDU
  tp_begin_dialogue_rc (*! constructor *) :
    tp_begin_dialogue_result,
    tp_begin_channel_diagnostic_Opt,
    nat (*correlator*)  ->
      PDU
  tp_bid_ri (*! constructor *) :
    bool (*ccr_token_requested*),
    correlator_Opt (*last_partner_identifier*)      ->
      PDU
  tp_bid_rc (*! constructor *) : tp_bid_result -> PDU
  tp_end_dialogue_ri (*! constructor *) : bool (*tp_end_dialogue_confirmation*)   -> PDU
  tp_end_dialogue_rc (*! constructor *) : -> PDU
  tp_U_error_ri (*! constructor *) : -> PDU
  tp_U_error_rc (*! constructor *) : -> PDU
  tp_abort_ri (*! constructor *) : tp_abort_type, PDUqueue (*user_data*)  -> PDU
  tp_abort_ri (*! constructor *) : tp_abort_type, tp_P_abort_diagnostic -> PDU
  tp_grant_control_ri (*! constructor *) : -> PDU
  tp_request_control_ri (*! constructor *) : -> PDU
  tp_handshake_ri (*! constructor *) : confirmation_urgency_Opt -> PDU
  tp_handshake_rc (*! constructor *) : -> PDU
  tp_handshake_and_grant_control_ri (*! constructor *) : confirmation_urgency -> PDU
  tp_handshake_and_grant_control_rc (*! constructor *) : -> PDU
  tp_defer_ri (*! constructor *) : tp_defer_type -> PDU
  tp_prepare_ri (*! constructor *) : bool_Opt (*data_permitted_Opt*) -> PDU
  tp_heuristic_report_ri (*! constructor *) : heuristic_report -> PDU
  tp_token_please_ri (*! constructor *) : -> PDU
  tp_token_give_ri (*! constructor *) : reason, correlator_Opt -> PDU
  tp_recover_ri (*! constructor *) : recovery_context_handle -> PDU
  tp_initialize_ri (*! constructor *) :
    protocol_version,
    bool (*contention_winner_assignment*),
    bool (*bid_mandatory*),
    recovery_context_handle_Opt ->
      PDU
  tp_initialize_rc (*! constructor *) :
    protocol_version,
    recovery_context_handle_Opt,
    tp_assoc_estab_diagnostic_Opt ->
      PDU
  tp_begin_dialogue_ri_pdu,
  tp_begin_dialogue_rc_pdu,
  tp_bid_ri_pdu,
  tp_bid_rc_pdu,
  tp_end_dialogue_ri_pdu,
  tp_end_dialogue_rc_pdu,
  tp_U_error_ri_pdu,
  tp_U_error_rc_pdu,
  tp_abort_ri_pdu,
  tp_grant_control_ri_pdu,
  tp_request_control_ri_pdu,
  tp_handshake_ri_pdu,
  tp_handshake_rc_pdu,
  tp_handshake_and_grant_control_ri_pdu,
  tp_handshake_and_grant_control_rc_pdu,
  tp_defer_ri_pdu,
  tp_prepare_ri_pdu,
  tp_heuristic_report_ri_pdu,
  tp_token_please_ri_pdu,
  tp_token_give_ri_pdu,
  tp_recover_ri_pdu,
  tp_initialize_ri_pdu,
  tp_initialize_rc_pdu,
  first_TPpdu (*! constructor *),
  last_TPpdu : ->
      PDUkey
  Istp_begin_dialogue_ri,
  Istp_begin_dialogue_rc,
  Istp_bid_ri,
  Istp_bid_rc,
  Istp_end_dialogue_ri,
  Istp_end_dialogue_rc,
  Istp_U_error_ri,
  Istp_U_error_rc,
  Istp_abort_ri,
  Istp_grant_control_ri,
  Istp_request_control_ri,
  Istp_handshake_ri,
  Istp_handshake_rc,
  Istp_handshake_and_grant_control_ri,
  Istp_handshake_and_grant_control_rc,
  Istp_defer_ri,
  Istp_prepare_ri,
  Istp_heuristic_report_ri,
  Istp_token_please_ri,
  Istp_token_give_ri,
  Istp_recover_ri,
  Istp_initialize_ri,
  Istp_initialize_rc,
  is_dialogue,
  is_channel,
  IsTPpdu :
    PDU -> bool
  get_type : PDU -> tp_abort_type
  get_aedO : PDU -> tp_assoc_estab_diagnostic_Opt
  get_bcdO : PDU -> tp_begin_channel_diagnostic_Opt
  get_bdc : PDU -> tp_begin_dialogue_confirmation
  get_bddO : PDU -> tp_begin_dialogue_diagnostic_Opt
  get_bdr : PDU -> tp_begin_dialogue_result
  get_bid : PDU -> bool (*bid_mandatory*)
  get_br : PDU -> tp_bid_result
  get_btO : PDU -> bool_Opt (*begin_transaction_Opt*)
  get_cnfu : PDU -> confirmation_urgency
  get_cnfuO : PDU -> confirmation_urgency_Opt
```

```
  get_ctr : PDU -> bool (*ccr_token_requested*)
  get_cu : PDU -> channel_utilization
  get_cwa : PDU -> bool (*contention_winner_assignment*)
  get_dcc : PDU -> nat (*correlator*)
  get_dccO : PDU -> correlator_Opt
  get_dpO : PDU -> bool_Opt (*data_permitted_Opt*)
  get_dt : PDU -> tp_defer_type
  get_edc : PDU -> bool (*tp_end_dialogue_confirmation*)
  get_hr : PDU -> heuristic_report
  get_ittO : PDU -> (*initiating*) TPSU_title_Opt
  get_lpiO : PDU -> correlator_Opt (*last_partner_identifier*)
  get_pad : PDU -> tp_P_abort_diagnostic
  get_pv : PDU -> protocol_version
  get_rch : PDU -> recovery_context_handle
  get_rchO : PDU -> recovery_context_handle_Opt
  get_rttO : PDU -> (*recipient*) TPSU_title_Opt
  get_sfu : PDU -> functional_units
  get_fuO : PDU -> functional_units_Opt
  get_reason : PDU -> reason
  get_ud : PDU -> PDUqueue (*user_data*)
eqns
    forall
      aedO :
      tp_assoc_estab_diagnostic_Opt,
      bcdO : tp_begin_channel_diagnostic_Opt,
      bdc : tp_begin_dialogue_confirmation,
      bddO : tp_begin_dialogue_diagnostic_Opt,
      bdr : tp_begin_dialogue_result,
      bid : bool (*bid_mandatory*),
      br : tp_bid_result,
      btO : bool_Opt (*begin_transaction_Opt*),
      cnfu : confirmation_urgency,
      cnfuO : confirmation_urgency_Opt,
      ctr : bool (*ccr_token_requested*),
      cu : channel_utilization,
      cwa : bool (*contention_winner_assignment*),
      dcc : nat (*correlator*),
      dccO : correlator_Opt,
      dpO : bool_Opt (*data_permitted_Opt*),
      dt : tp_defer_type,
      edc : bool (*tp_end_dialogue_confirmation*),
      hr : heuristic_report,
      ittO :  (*initiating*) TPSU_title_Opt,
      lpiO : correlator_Opt (*last_partner_identifier*),
      pad : tp_P_abort_diagnostic,
      pv : protocol_version,
      rch : recovery_context_handle,
      rchO : recovery_context_handle_Opt,
      rttO :  (*recipient*) TPSU_title_Opt,
      sfu : functional_units,
      fuO : functional_units_Opt,
      tokr : reason,
      ud : PDUqueue (*user_data*),
      pdu : PDU
```

```
ofsort PDUkey
  tp_begin_dialogue_ri_pdu = first_TPpdu;
  tp_begin_dialogue_rc_pdu = succ(tp_begin_dialogue_ri_pdu);
  tp_bid_ri_pdu = succ(tp_begin_dialogue_rc_pdu);
  tp_bid_rc_pdu = succ(tp_bid_ri_pdu);
  tp_end_dialogue_ri_pdu = succ(tp_bid_rc_pdu);
  tp_end_dialogue_rc_pdu = succ(tp_end_dialogue_ri_pdu);
  tp_U_error_ri_pdu = succ(tp_end_dialogue_rc_pdu);
  tp_U_error_rc_pdu = succ(tp_U_error_ri_pdu);
  tp_abort_ri_pdu = succ(tp_U_error_rc_pdu);
  tp_grant_control_ri_pdu = succ(tp_abort_ri_pdu);
  tp_request_control_ri_pdu = succ(tp_grant_control_ri_pdu);
  tp_handshake_ri_pdu = succ(tp_request_control_ri_pdu);
  tp_handshake_rc_pdu = succ(tp_handshake_ri_pdu);
  tp_handshake_and_grant_control_ri_pdu = succ(tp_handshake_rc_pdu);
  tp_handshake_and_grant_control_rc_pdu =
    succ(tp_handshake_and_grant_control_ri_pdu);
  tp_defer_ri_pdu = succ(tp_handshake_and_grant_control_rc_pdu);
  tp_prepare_ri_pdu = succ(tp_defer_ri_pdu);
  tp_heuristic_report_ri_pdu = succ(tp_prepare_ri_pdu);
  tp_token_please_ri_pdu = succ(tp_heuristic_report_ri_pdu);
  tp_token_give_ri_pdu = succ(tp_token_please_ri_pdu);
  tp_recover_ri_pdu = succ(tp_token_give_ri_pdu);
  tp_initialize_ri_pdu = succ(tp_recover_ri_pdu);
  tp_initialize_rc_pdu = succ(tp_initialize_ri_pdu);
  last_TPpdu = tp_initialize_rc_pdu;
  key(tp_begin_dialogue_ri(itt0, rtt0, sfu, bt0, bdc, dcc, lpi0, ud)) =
    tp_begin_dialogue_ri_pdu;
  key(tp_begin_dialogue_ri(sfu, dcc, cu, lpi0)) = tp_begin_dialogue_ri_pdu;
  key(tp_begin_dialogue_rc(fu0, bdr, bdd0, dcc, ud)) =
    tp_begin_dialogue_rc_pdu;
  key(tp_begin_dialogue_rc(bdr, bcd0, dcc)) = tp_begin_dialogue_rc_pdu;
  key(tp_bid_ri(ctr, lpi0)) = tp_bid_ri_pdu;
  key(tp_bid_rc(br)) = tp_bid_rc_pdu;
  key(tp_end_dialogue_ri(edc)) = tp_end_dialogue_ri_pdu;
  key(tp_end_dialogue_rc) = tp_end_dialogue_rc_pdu;
  key(tp_U_error_ri) = tp_U_error_ri_pdu;
  key(tp_U_error_rc) = tp_U_error_rc_pdu;
  key(tp_abort_ri(user, ud)) = tp_abort_ri_pdu;
  key(tp_abort_ri(provider, pad)) = tp_abort_ri_pdu;
  key(tp_grant_control_ri) = tp_grant_control_ri_pdu;
  key(tp_request_control_ri) = tp_request_control_ri_pdu;
  key(tp_handshake_ri(cnfu0)) = tp_handshake_ri_pdu;
  key(tp_handshake_rc) = tp_handshake_rc_pdu;
  key(tp_handshake_and_grant_control_ri(cnfu)) =
    tp_handshake_and_grant_control_ri_pdu;
  key(tp_handshake_and_grant_control_rc) =
    tp_handshake_and_grant_control_rc_pdu;
  key(tp_defer_ri(dt)) = tp_defer_ri_pdu;
  key(tp_prepare_ri(dp0)) = tp_prepare_ri_pdu;
  key(tp_heuristic_report_ri(hr)) = tp_heuristic_report_ri_pdu;
  key(tp_token_please_ri) = tp_token_please_ri_pdu;
  key(tp_token_give_ri(tokr, dcc0)) = tp_token_give_ri_pdu;
  key(tp_recover_ri(rch)) = tp_recover_ri_pdu;
```

```
    key(tp_initialize_ri(pv, cwa, bid, rch0)) = tp_initialize_ri_pdu;
    key(tp_initialize_rc(pv, rch0, aed0)) = tp_initialize_rc_pdu;
  ofsort bool
    Istp_begin_dialogue_ri(pdu) = pdu eq tp_begin_dialogue_ri_pdu;
    Istp_begin_dialogue_rc(pdu) = pdu eq tp_begin_dialogue_rc_pdu;
    Istp_bid_ri(pdu) = pdu eq tp_bid_ri_pdu;
    Istp_bid_rc(pdu) = pdu eq tp_bid_rc_pdu;
    Istp_end_dialogue_ri(pdu) = pdu eq tp_end_dialogue_ri_pdu;
    Istp_end_dialogue_rc(pdu) = pdu eq tp_end_dialogue_rc_pdu;
    Istp_U_error_ri(pdu) = pdu eq tp_U_error_ri_pdu;
    Istp_U_error_rc(pdu) = pdu eq tp_U_error_rc_pdu;
    Istp_abort_ri(pdu) = pdu eq tp_abort_ri_pdu;
    Istp_grant_control_ri(pdu) = pdu eq tp_grant_control_ri_pdu;
    Istp_request_control_ri(pdu) = pdu eq tp_request_control_ri_pdu;
    Istp_handshake_ri(pdu) = pdu eq tp_handshake_ri_pdu;
    Istp_handshake_rc(pdu) = pdu eq tp_handshake_rc_pdu;
    Istp_handshake_and_grant_control_ri(pdu) =
      pdu eq tp_handshake_and_grant_control_ri_pdu;
    Istp_handshake_and_grant_control_rc(pdu) =
      pdu eq tp_handshake_and_grant_control_rc_pdu;
    Istp_defer_ri(pdu) = pdu eq tp_defer_ri_pdu;
    Istp_prepare_ri(pdu) = pdu eq tp_prepare_ri_pdu;
    Istp_heuristic_report_ri(pdu) = pdu eq tp_heuristic_report_ri_pdu;
    Istp_token_please_ri(pdu) = pdu eq tp_token_please_ri_pdu;
    Istp_token_give_ri(pdu) = pdu eq tp_token_give_ri_pdu;
    Istp_recover_ri(pdu) = pdu eq tp_recover_ri_pdu;
    Istp_initialize_ri(pdu) = pdu eq tp_initialize_ri_pdu;
    Istp_initialize_rc(pdu) = pdu eq tp_initialize_rc_pdu;
    IsTPpdu(pdu) = (first_TPpdu le key(pdu)) and (key(pdu) le last_TPpdu);
    is_dialogue(tp_begin_dialogue_ri(itt0, rtt0, sfu, bt0, bdc, dcc, lpi0, ud))
    =
      true;
    is_dialogue(tp_begin_dialogue_ri(sfu, dcc, cu, lpi0)) = false;
    is_dialogue(tp_begin_dialogue_rc(fu0, bdr, bdd0, dcc, ud)) = true;
    is_dialogue(tp_begin_dialogue_rc(bdr, bcd0, dcc)) = false;
    is_channel(tp_begin_dialogue_ri(itt0, rtt0, sfu, bt0, bdc, dcc, lpi0, ud)) =
      false;
    is_channel(tp_begin_dialogue_ri(sfu, dcc, cu, lpi0)) = true;
    is_channel(tp_begin_dialogue_rc(fu0, bdr, bdd0, dcc, ud)) = false;
    is_channel(tp_begin_dialogue_rc(bdr, bcd0, dcc)) = true;
  ofsort tp_abort_type
    get_type(tp_abort_ri(user, ud)) = user;
    get_type(tp_abort_ri(provider, pad)) = provider;
  ofsort tp_assoc_estab_diagnostic_Opt
    get_aed0(tp_initialize_rc(pv, rch0, aed0)) = aed0;
  ofsort tp_begin_channel_diagnostic_Opt
    get_bcd0(tp_begin_dialogue_rc(bdr, bcd0, dcc)) = bcd0;
  ofsort tp_begin_dialogue_confirmation
    get_bdc(tp_begin_dialogue_ri(itt0, rtt0, sfu, bt0, bdc, dcc, lpi0, ud)) =
      bdc;
  ofsort tp_begin_dialogue_diagnostic_Opt
    get_bdd0(tp_begin_dialogue_rc(fu0, bdr, bdd0, dcc, ud)) = bdd0;
  ofsort tp_begin_dialogue_result
    get_bdr(tp_begin_dialogue_rc(fu0, bdr, bdd0, dcc, ud)) = bdr;
    get_bdr(tp_begin_dialogue_rc(bdr, bcd0, dcc)) = bdr;
  ofsort bool (*bid_mandatory*)
    get_bid(
      tp_initialize_ri(pv, cwa, bid, rch0)) =
      bid;
  ofsort tp_bid_result get_br(tp_bid_rc(br)) = br;
  ofsort bool_Opt (*begin_transaction_Opt*)
    get_bt0(
      tp_begin_dialogue_ri(itt0, rtt0, sfu, bt0, bdc, dcc, lpi0, ud)) =
      bt0;
  ofsort confirmation_urgency
    get_cnfu(tp_handshake_and_grant_control_ri(cnfu)) = cnfu;
  ofsort confirmation_urgency_Opt get_cnfu0(tp_handshake_ri(cnfu0)) = cnfu0;
  ofsort bool (*ccr_token_requested*)
    get_ctr(
      tp_bid_ri(ctr, lpi0)) =
      ctr;
  ofsort channel_utilization
    get_cu(tp_begin_dialogue_ri(sfu, dcc, cu, lpi0)) = cu;
  ofsort bool (*contention_winner_assignment*)
    get_cwa(
      tp_initialize_ri(pv, cwa, bid, rch0)) =
      cwa;
  ofsort nat (*correlator*)
    get_dcc(
      tp_begin_dialogue_ri(itt0, rtt0, sfu, bt0, bdc, dcc, lpi0, ud)) =
      dcc;
    get_dcc(tp_begin_dialogue_ri(sfu, dcc, cu, lpi0)) = dcc;
    get_dcc(tp_begin_dialogue_rc(fu0, bdr, bdd0, dcc, ud)) = dcc;
    get_dcc(tp_begin_dialogue_rc(bdr, bcd0, dcc)) = dcc;
  ofsort correlator_Opt get_dcc0(tp_token_give_ri(tokr, dcc0)) = dcc0;
  ofsort reason get_reason(tp_token_give_ri(tokr, dcc0)) = tokr;
  ofsort bool_Opt (*data_permitted_Opt*)
    get_dp0(
      tp_prepare_ri(dp0)) =
      dp0;
  ofsort tp_defer_type get_dt(tp_defer_ri(dt)) = dt;
  ofsort bool (*tp_end_dialogue_confirmation*)
    get_edc(
      tp_end_dialogue_ri(edc)) =
      edc;
  ofsort heuristic_report get_hr(tp_heuristic_report_ri(hr)) = hr;
  ofsort  (*initiating*) TPSU_title_Opt
    get_itt0(tp_begin_dialogue_ri(itt0, rtt0, sfu, bt0, bdc, dcc, lpi0, ud)) =
      itt0;
  ofsort correlator_Opt (*last_partner_identifier*)
    get_lpi0(
      tp_begin_dialogue_ri(itt0, rtt0, sfu, bt0, bdc, dcc, lpi0, ud)) =
      lpi0;
    get_lpi0(tp_begin_dialogue_ri(sfu, dcc, cu, lpi0)) = lpi0;
    get_lpi0(tp_bid_ri(ctr, lpi0)) = lpi0;
  ofsort tp_P_abort_diagnostic get_pad(tp_abort_ri(provider, pad)) = pad;
  ofsort protocol_version
    get_pv(tp_initialize_ri(pv, cwa, bid, rch0)) = pv;
    get_pv(tp_initialize_rc(pv, rchO, aedO)) = pv;
  ofsort recovery_context_handle get_rch(tp_recover_ri(rch)) = rch;
  ofsort recovery_context_handle_Opt
    get_rchO(tp_initialize_ri(pv, cwa, bid, rchO)) = rchO;
    get_rchO(tp_initialize_rc(pv, rchO, aedO)) = rchO;
  ofsort  (*recipient*) TPSU_title_Opt
    get_rttO(tp_begin_dialogue_ri(ittO, rttO, sfu, btO, bdc, dcc, lpiO, ud)) =
      rttO;
  ofsort functional_units
    get_sfu(tp_begin_dialogue_ri(ittO, rttO, sfu, btO, bdc, dcc, lpiO, ud)) =
      sfu;
    get_sfu(tp_begin_dialogue_ri(sfu, dcc, cu, lpiO)) = sfu;
  ofsort functional_units_Opt
    get_fuO(tp_begin_dialogue_rc(fuO, bdr, bddO, dcc, ud)) = fuO;
  ofsort PDUqueue (*user_data*)
    get_ud(
      tp_begin_dialogue_ri(ittO, rttO, sfu, btO, bdc, dcc, lpiO, ud)) =
      ud;
    get_ud(tp_begin_dialogue_rc(fuO, bdr, bddO, dcc, ud)) = ud;
    get_ud(tp_abort_ri(user, ud)) = ud;
endtype
(* -------------------------------------------------------------------------

module TP_PDU is parameters, BasicPDU

    type PDU is
        tp_begin_dialogue_ri  ( ITTO: TPSU_title_Opt,
                                RTTO: TPSU_title_Opt,
                                SFU: functional_units,
                                BTO: bool_Opt,
                                BDC: tp_begin_dialogue_confirmation,
                                DCC: nat,
                                LPIO: correlator_Opt,
                                UD: PDUqueue)
    |   tp_begin_dialogue_ri  ( SFU: functional_units,
                                DCC: nat,
                                CU: channel_utilization,
                                LPIO: correlator_Opt)
    |   tp_begin_dialogue_rc  ( FUO: functional_units_Opt,
                                BDR: tp_begin_dialogue_result,
                                BDDO: tp_begin_dialogue_diagnostic_Opt,
                                DCC: nat,
                                UD: PDUqueue)
    |   tp_begin_dialogue_rc  ( BDR: tp_begin_dialogue_result,
                                BCDO: tp_begin_channel_diagnostic_Opt,
                                DCC: nat)
    |   tp_bid_ri             ( CTR: bool,
                                LPIO: correlator_Opt)
    |   tp_bid_rc             ( BR: tp_bid_result)
    |   tp_end_dialogue_ri    ( EDC: bool)
    |   tp_end_dialogue_rc
    |   tp_U_error_ri
    |   tp_U_error_rc
    |   tp_abort_ri           ( AT: tp_abort_type,
                                UD: PDUqueue)
    |   tp_abort_ri           ( AT: tp_abort_type,
                                PAD: tp_P_abort_diagnostic)
    |   tp_grant_control_ri
    |   tp_request_control_ri
    |   tp_handshake_ri       ( CNFUO: confirmation_urgency_Opt)
    |   tp_handshake_rc
    |   tp_handshake_and_grant_control_ri ( CNFU: confirmation_urgency)
    |   tp_handshake_and_grant_control_rc
    |   tp_defer_ri           ( DT: tp_defer_type)
    |   tp_prepare_ri         ( DPO: bool_Opt)
    |   tp_heuristic_report_ri ( HR: heuristic_report)
    |   tp_token_please_ri
    |   tp_token_give_ri      ( TOKR: reason,
                                DCCO: correlator_Opt)
    |   tp_recover_ri         ( RCH: recovery_context_handle)
    |   tp_initialize_ri      ( PV: protocol_version,
                                CWA: bool,
                                BID: bool,
                                RCHO: recovery_context_handle_Opt)
    |   tp_initialize_rc      ( PV: protocol_version,
                                RCHO: recovery_context_handle_Opt,
                                AEDO: tp_assoc_estab_diagnostic_Opt)
    endtype

/* no traduction is needed for tp_*_pdu */

function tp_begin_dialogue_ri_pdu : PDUkey is first_TPpdu endfunc
function tp_begin_dialogue_rc_pdu : PDUkey is succ(tp_begin_dialogue_ri_pdu) endfunc
function tp_bid_ri_pdu : PDUkey is succ(tp_begin_dialogue_rc_pdu) endfunc
function tp_bid_rc_pdu : PDUkey is succ(tp_bid_ri_pdu) endfunc
function tp_end_dialogue_ri_pdu : PDUkey is succ(tp_bid_rc_pdu) endfunc
function tp_end_dialogue_rc_pdu : PDUkey is succ(tp_end_dialogue_ri_pdu) endfunc
function tp_U_error_ri_pdu : PDUkey is succ(tp_end_dialogue_rc_pdu) endfunc
function tp_U_error_rc_pdu : PDUkey is succ(tp_U_error_ri_pdu) endfunc
function tp_abort_ri_pdu : PDUkey is succ(tp_U_error_rc_pdu) endfunc
function tp_grant_control_ri_pdu : PDUkey is succ(tp_abort_ri_pdu) endfunc
function tp_request_control_ri_pdu : PDUkey is succ(tp_grant_control_ri_pdu) endfunc
function tp_handshake_ri_pdu : PDUkey is succ(tp_request_control_ri_pdu) endfunc
function tp_handshake_rc_pdu : PDUkey is succ(tp_handshake_ri_pdu) endfunc
function tp_handshake_and_grant_control_ri_pdu : PDUkey is succ(tp_handshake_rc_pdu) endfunc
function tp_handshake_and_grant_control_rc_pdu : PDUkey is
        succ(tp_handshake_and_grant_control_ri_pdu) endfunc
function tp_defer_ri_pdu : PDUkey is succ(tp_handshake_and_grant_control_rc_pdu) endfunc
function tp_prepare_ri_pdu : PDUkey is succ(tp_defer_ri_pdu) endfunc
function tp_heuristic_report_ri_pdu : PDUkey is succ(tp_prepare_ri_pdu) endfunc
function tp_token_please_ri_pdu : PDUkey is succ(tp_heuristic_report_ri_pdu) endfunc
function tp_token_give_ri_pdu : PDUkey is succ(tp_token_please_ri_pdu) endfunc
function tp_recover_ri_pdu : PDUkey is succ(tp_token_give_ri_pdu) endfunc
function tp_initialize_ri_pdu : PDUkey is succ(tp_recover_ri_pdu) endfunc
function tp_initialize_rc_pdu : PDUkey is succ(tp_initialize_ri_pdu) endfunc
function last_TPpdu : PDUkey is tp_initialize_rc_pdu endfunc

/* no translation is needed for key */
```

```
function key (P: PDU) : PDUkey is
    case P in
        tp_begin_dialogue_ri(ITTO: TPSU_title_Opt,...)
                -> tp_begin_dialogue_ri_pdu
    |   tp_begin_dialogue_ri(SFU: functional_units,...)
                -> tp_begin_dialogue_ri_pdu
    |   tp_begin_dialogue_rc(FUO: functional_units_Opt,...)
                -> tp_begin_dialogue_rc_pdu
    |   tp_begin_dialogue_rc(...)
                -> tp_begin_dialogue_rc_pdu
    |   tp_bid_ri(...)
                -> tp_bid_ri_pdu
    |   tp_bid_rc(...)
                -> tp_bid_rc_pdu
    |   tp_end_dialogue_ri(...)
                -> tp_end_dialogue_ri_pdu
    |   tp_end_dialogue_rc
                -> tp_end_dialogue_rc_pdu
    |   tp_U_error_ri
                -> tp_U_error_ri_pdu
    |   tp_U_error_rc
                -> tp_U_error_rc_pdu
    |   tp_abort_ri(UD: PDUqueue,...)
                -> tp_abort_ri_pdu
    |   tp_abort_ri(PAD: tp_P_abort_diagnostic,...)
                -> tp_abort_ri_pdu
    |   tp_grant_control_ri
                -> tp_grant_control_ri_pdu
    |   tp_request_control_ri
                -> tp_request_control_ri_pdu
    |   tp_handshake_ri(...)
                -> tp_handshake_ri_pdu
    |   tp_handshake_rc
                -> tp_handshake_rc_pdu
    |   tp_handshake_and_grant_control_ri(...)
                -> tp_handshake_and_grant_control_ri_pdu
    |   tp_handshake_and_grant_control_rc
                -> tp_handshake_and_grant_control_rc_pdu
    |   tp_defer_ri(...)
                -> tp_defer_ri_pdu
    |   tp_prepare_ri(...)
                -> tp_prepare_ri_pdu
    |   tp_heuristic_report_ri(...)
                -> tp_heuristic_report_ri_pdu
    |   tp_token_please_ri
                -> tp_token_please_ri_pdu
    |   tp_token_give_ri(...)
                -> tp_token_give_ri_pdu
    |   tp_recover_ri(...)
                -> tp_recover_ri_pdu
    |   tp_initialize_ri(...)
                -> tp_initialize_ri_pdu
    |   tp_initialize_rc(...)
                -> tp_initialize_rc_pdu
    endcase
endfunc

    function Istp_begin_dialogue_ri (P: PDU) : bool is
        P match tp_begin_dialogue_ri (...)
        /* the both constructors match ! */
    endfunc

    function Istp_begin_dialogue_rc (P: PDU) : bool is
        P match tp_begin_dialogue_rc (...)
        /* the both constructors match ! */
    endfunc

    function Istp_bid_ri (P: PDU) : bool is P match tp_bid_ri (...) endfunc

    function Istp_bid_rc (P: PDU) : bool is P match tp_bid_rc (...) endfunc

    function Istp_end_dialogue_ri (P: PDU) : bool is P match tp_end_dialogue_ri (...) endfunc

    function Istp_end_dialogue_rc (P: PDU) : bool is P match tp_end_dialogue_rc (...) endfunc

    function Istp_U_error_ri (P: PDU) : bool is P match tp_U_error_ri (...) endfunc

    function Istp_U_error_rc (P: PDU) : bool is P match tp_U_error_rc (...) endfunc

    function Istp_abort_ri (P: PDU) : bool is P match tp_abort_ri (...) endfunc

    function Istp_grant_control_ri (P: PDU) : bool is P match tp_grant_control_ri endfunc

    function Istp_request_control_ri (P: PDU) : bool is P match tp_request_control_ri endfunc

    function Istp_handshake_ri (P: PDU) : bool is P match tp_handshake_ri (...) endfunc

    function Istp_handshake_rc (P: PDU) : bool is P match tp_handshake_rc endfunc

    function Istp_handshake_and_grant_control_ri (P: PDU) : bool is
        P match tp_handshake_and_grant_control_ri (...) endfunc

    function Istp_handshake_and_grant_control_rc (P: PDU) : bool is
        P match tp_handshake_and_grant_control_rc endfunc

    function Istp_defer_ri (P: PDU) : bool is P match tp_defer_ri (...) endfunc

    function Istp_prepare_ri (P: PDU) : bool is P match tp_prepare_ri (...) endfunc

    function Istp_heuristic_report_ri (P: PDU) : bool is P match tp_heuristic_report_ri (...) endfunc

    function Istp_token_please_ri (P: PDU) : bool is P match tp_token_please_ri endfunc

    function Istp_token_give_ri (P: PDU) : bool is P match tp_token_give_ri (...) endfunc

    function Istp_recover_ri (P: PDU) : bool is P match tp_recover_ri (...) endfunc

    function Istp_initialize_ri (P: PDU) : bool is P match tp_initialize_ri (...) endfunc
    function Istp_initialize_rc (P: PDU) : bool is P match tp_initialize_rc (...) endfunc

    function is_dialogue (P: PDU) : bool is
        (P match tp_begin_dialogue_ri (ITTO: TPSU_title_Opt,...)) or
        (P match tp_begin_dialogue_rc (FUO: functional_units_Opt,...))
    endfunc

    function is_channel (P: PDU) : bool is
        (P match tp_begin_dialogue_ri (CU: channel_utilization,...)) or
        (P match tp_begin_dialogue_rc (BCDO: tp_begin_channel_diagnostic_Opt))
    endfunc

    function IsTPpdu  (P: PDU) : bool is
/* direct translation
        (first_TPpdu le key(pdu)) and (key(pdu) le last_TPpdu) */
        P match  any PDU
    endfunc

    function get_type (P: PDU) : tp_abort_type is select AT in P endfunc

    function get_aedO (P: PDU) : tp_assoc_estab_diagnostic_Opt is select AEDO in P endfunc

    function get_bcdO (P: PDU) : tp_begin_channel_diagnostic_Opt is select BCDO in P endfunc

    function get_bdc (P: PDU) : tp_begin_dialogue_confirmation is select BDC in P endfunc

    function get_bddO (P: PDU) : tp_begin_dialogue_diagnostic_Opt is select BDDO in P endfunc

    function get_bdr (P: PDU) : tp_begin_dialogue_result is select BDR in P endfunc

    function get_bid (P: PDU) : bool is select BID in P endfunc

    function get_br (P: PDU) : tp_bid_result is select BR in P endfunc

    function get_btO (P: PDU) : bool_Opt is select BTO in P endfunc

    function get_cnfu (P: PDU) : confirmation_urgency is
        case P in
            tp_handshake_and_grant_control_ri (CNFU:=C confirmation_urgency)
                -> C
        /* the following equation is shifted from PDU type */
        |   tp_handshake_ri (CNFUO:=CO : confirmation_urgency_Opt)
                when not (is_absent (CO))
                -> get_cnfu (CO)
    endfunc

    function get_cnfuO (P: PDU) : confirmation_urgency_Opt is select CNFUO in P endfunc

    function get_ctr (P: PDU) : bool is select CTR in P endfunc

    function get_cu (P: PDU) : channel_utilization is select CU in P endfunc

    function get_cwa (P: PDU) : bool is select CWA in P endfunc
    function get_dcc (P: PDU) : nat is select DCC in P endfunc

    function get_dccO (P: PDU) : correlator_Opt is select DCCO in P endfunc

    function get_reason (P: PDU) : reason is select TOKR in P endfunc

    function get_dpO (P: PDU) : bool_Opt is select DPO in P endfunc

    function get_dt (P: PDU) : tp_defer_type is select DT in P endfunc

    function get_edc (P: PDU) : bool is select EDC in P endfunc

    function get_hr (P: PDU) : heuristic_report is select HR in P endfunc

    function get_ittO (P: PDU) : TPSU_title_Opt is select ITTO in P endfunc

    function get_lpiO (P: PDU) : correlator_Opt is select LPIO in P endfunc

    function get_pad (P: PDU) : tp_P_abort_diagnostic is select PAD in P endfunc

    function get_pv (P: PDU) : protocol_version is select PV in P endfunc

    function get_rch (P: PDU) : recovery_context_handle is select RCH in P endfunc

    function get_rchO (P: PDU) : recovery_context_handle_Opt is select RCHO in P endfunc

    function get_rttO (P: PDU) : TPSU_title_Opt is select RTTO in P endfunc

    function get_sfu (P: PDU) : functional_units is select SFU in P endfunc

    function get_fuO (P: PDU) : functional_units_Opt is select FUO in P endfunc

    function get_ud (P: PDU) : PDUqueue is select UD in P endfunc
endmod
========================================================================= *)

type CCR_PDU is TP_PDU
opns
  c_begin_ri_pdu,
  c_begin_rc_pdu,
  c_prepare_ri_pdu,
  c_ready_ri_pdu,
  c_commit_ri_pdu,
  c_commit_rc_pdu,
  c_rollback_ri_pdu,
  c_rollback_rc_pdu,
  c_recover_ri_pdu,
  c_recover_rc_pdu,
  first_CCRpdu,
  last_CCRpdu : ->
      PDUkey
  Isc_begin_ri,
  Isc_begin_rc,
  Isc_prepare_ri,
  Isc_ready_ri,
  Isc_commit_ri,
  Isc_commit_rc,
  Isc_rollback_ri,
  Isc_rollback_rc,
  Isc_recover_ri,
  Isc_recover_rc,
  IsCCRpdu :
    PDU -> bool
eqns
    forall pdu : PDU
  ofsort PDUkey
    first_CCRpdu = succ(last_TPpdu);
    c_begin_ri_pdu = first_CCRpdu;
    c_begin_rc_pdu = succ(c_begin_ri_pdu);
    c_prepare_ri_pdu = succ(c_begin_rc_pdu);
    c_ready_ri_pdu = succ(c_prepare_ri_pdu);
    c_commit_ri_pdu = succ(c_ready_ri_pdu);
    c_commit_rc_pdu = succ(c_commit_ri_pdu);
    c_rollback_ri_pdu = succ(c_commit_rc_pdu);
    c_rollback_rc_pdu = succ(c_rollback_ri_pdu);
    c_recover_ri_pdu = succ(c_rollback_rc_pdu);
    c_recover_rc_pdu = succ(c_recover_ri_pdu);
    last_CCRpdu = c_recover_rc_pdu;
  ofsort bool
    Isc_begin_ri(pdu) = pdu eq c_begin_ri_pdu;
    Isc_begin_rc(pdu) = pdu eq c_begin_rc_pdu;
    Isc_prepare_ri(pdu) = pdu eq c_prepare_ri_pdu;
    Isc_ready_ri(pdu) = pdu eq c_ready_ri_pdu;
    Isc_commit_ri(pdu) = pdu eq c_commit_ri_pdu;
    Isc_commit_rc(pdu) = pdu eq c_commit_rc_pdu;
    Isc_rollback_ri(pdu) = pdu eq c_rollback_ri_pdu;
    Isc_rollback_rc(pdu) = pdu eq c_rollback_rc_pdu;
    Isc_recover_ri(pdu) = pdu eq c_recover_ri_pdu;
    Isc_recover_rc(pdu) = pdu eq c_recover_rc_pdu;
    IsCCRpdu(pdu) = (first_CCRpdu le key(pdu)) and (key(pdu) le last_CCRpdu);
endtype
(* ---------------------------------------------------------------------------

module CCR_PDU is TP_PDU
  Isc_begin_ri,
  Isc_begin_rc,
  Isc_prepare_ri,
  Isc_ready_ri,
  Isc_commit_ri,
  Isc_commit_rc,
  Isc_rollback_ri,
  Isc_rollback_rc,
  Isc_recover_ri,
  Isc_recover_rc,
  IsCCRpdu :
    PDU -> bool
eqns
```

```
    forall pdu : PDU
/* not translation is needed for functions
  c_begin_ri_pdu,
  c_begin_rc_pdu,
  c_prepare_ri_pdu,
  c_ready_ri_pdu,
  c_commit_ri_pdu,
  c_commit_rc_pdu,
  c_rollback_ri_pdu,
  c_rollback_rc_pdu,
  c_recover_ri_pdu,
  c_recover_rc_pdu,
  first_CCRpdu,
  last_CCRpdu

*/
    function first_CCRpdu : PDUkey is succ(last_TPpdu) endfunc
    function c_begin_ri_pdu : PDUkey is first_CCRpdu endfunc
    function c_begin_rc_pdu : PDUkey is succ(c_begin_ri_pdu) endfunc
    function c_prepare_ri_pdu : PDUkey is succ(c_begin_rc_pdu) endfunc
    function c_ready_ri_pdu : PDUkey is succ(c_prepare_ri_pdu) endfunc
    function c_commit_ri_pdu : PDUkey is succ(c_ready_ri_pdu) endfunc
    function c_commit_rc_pdu : PDUkey is succ(c_commit_ri_pdu) endfunc
    function c_rollback_ri_pdu : PDUkey is succ(c_commit_rc_pdu) endfunc
    function c_rollback_rc_pdu : PDUkey is succ(c_rollback_ri_pdu) endfunc
    function c_recover_ri_pdu : PDUkey is succ(c_rollback_rc_pdu) endfunc
    function c_recover_rc_pdu : PDUkey is succ(c_recover_ri_pdu) endfunc
    function last_CCRpdu : PDUkey is c_recover_rc_pdu endfunc

/* no value of PDU is mapped in c_*_pdu by key -> error ! */

    function Isc_begin_ri (P: PDU) : bool is P eq c_begin_ri_pdu endfunc
    function Isc_begin_rc (P: PDU) : bool is P eq c_begin_rc_pdu endfunc
    function Isc_prepare_ri (P: PDU) : bool is P eq c_prepare_ri_pdu endfunc
    function Isc_ready_ri (P: PDU) : bool is P eq c_ready_ri_pdu endfunc
    function Isc_commit_ri (P: PDU) : bool is P eq c_commit_ri_pdu endfunc
    function Isc_commit_rc (P: PDU) : bool is P eq c_commit_rc_pdu endfunc
    function Isc_rollback_ri (P: PDU) : bool is P eq c_rollback_ri_pdu endfunc
    function Isc_rollback_rc (P: PDU) : bool is P eq c_rollback_rc_pdu endfunc
    function Isc_recover_ri (P: PDU) : bool is P eq c_recover_ri_pdu endfunc
    function Isc_recover_rc (P: PDU) : bool is P eq c_recover_rc_pdu endfunc
    function IsCCRpdu (P: PDU) : bool is (first_CCRpdu le key(P)) and (key(P) le last_CCRpdu) endfunc
endmod
=========================================================================== *)


type ACSE_PDU is CCR_PDU
opns
  AARQ_pdu, AARE_pdu, ABRT_pdu, ARLQ_pdu, ARLE_pdu, first_ACSEpdu, last_ACSEpdu
  : ->
      PDUkey
  IsAARQ, IsAARE, IsABRT, IsARLQ, IsARLE, IsACSEpdu : PDU -> bool
eqns
    forall pdu : PDU
  ofsort PDUkey
    first_ACSEpdu = succ(last_CCRpdu);
    AARQ_pdu = first_ACSEpdu;
    AARE_pdu = succ(AARQ_pdu);
    ABRT_pdu = succ(AARE_pdu);
    ARLQ_pdu = succ(ABRT_pdu);
    ARLE_pdu = succ(ARLQ_pdu);
    last_ACSEpdu = ARLE_pdu;
  ofsort bool
    IsAARQ(pdu) = pdu eq AARQ_pdu;
    IsAARE(pdu) = pdu eq AARE_pdu;
    IsABRT(pdu) = pdu eq ABRT_pdu;
    IsARLQ(pdu) = pdu eq ARLQ_pdu;
    IsARLE(pdu) = pdu eq ARLE_pdu;
    IsACSEpdu(pdu) = (first_ACSEpdu le key(pdu)) and (key(pdu) le last_ACSEpdu);
endtype

type UPDU is ACSE_PDU
opns updu : -> PDUkey Isupdu : PDU -> bool
eqns
    forall pdu : PDU
  ofsort PDUkey updu = Succ(last_ACSEpdu);
  ofsort bool IsUpdu(pdu) = pdu eq updu
endtype
(* ---------------------------------------------------------------------------

module ACSE_PDU is CCR_PDU

/* no translation is needed for
AARQ_pdu, AARE_pdu, ABRT_pdu, ARLQ_pdu, ARLE_pdu, first_ACSEpdu, last_ACSEpdu
functions */
    function first_ACSEpdu : PDUkey is succ(last_CCRpdu) endfunc
    function AARQ_pdu : PDUkey is first_ACSEpdu endfunc
    function AARE_pdu : PDUkey is succ(AARQ_pdu) endfunc
    function ABRT_pdu : PDUkey is succ(AARE_pdu) endfunc
    function ARLQ_pdu : PDUkey is succ(ABRT_pdu) endfunc
    function ARLE_pdu : PDUkey is succ(ARLQ_pdu) endfunc
    function last_ACSEpdu : PDUkey is ARLE_pdu endfunc

/* no PDUs are associeted by key to AARQ_pdu ... ARLE_pdu -->  error ! */

    function IsAARQ (P: PDU) : bool is P eq AARQ_pdu endfunc
    function IsAARE (P: PDU) : bool is P eq AARE_pdu endfunc
    function IsABRT (P: PDU) : bool is P eq ABRT_pdu endfunc
    function IsARLQ (P: PDU) : bool is P eq ARLQ_pdu endfunc
    function IsARLE (P: PDU) : bool is P eq ARLE_pdu endfunc
    function IsACSEpdu (P: PDU) : bool is
        (first_ACSEpdu le key(P)) and (key(P) le last_ACSEpdu) endfunc
endmod

module UPDU is ACSE_PDU
/* no translation is needed for updu */
    function updu : PDUkey is succ(last_ACSEpdu) endfunc
/* no PDUs are associeted by key to updu -->  error ! */
```

```
    function IsUpdu (P: PDU) : bool is P eq updu endfunc
endmod

module PDU is UPDU
opns
  Istp_D_begin_dialogue_ri,
  Istp_C_begin_dialogue_ri,
  Istp_D_begin_dialogue_rc,
  Istp_C_begin_dialogue_rc,
  Istp_begin_dialogue_rc_accepted,
  Istp_begin_dialogue_rc_rejected,
  is_handshake_ri,
  is_handshake_rc,
  has_cnfu :
    PDU -> bool
  is_user, is_provider : PDU -> bool
eqns
/* the following specifications could be writed using PM, too */
    function Istp_D_begin_dialogue_ri (P: PDU) : bool is
      Istp_begin_dialogue_ri(P) and is_dialogue(P) endfunc

    function Istp_C_begin_dialogue_ri (P: PDU) : bool is
      Istp_begin_dialogue_ri(P) and is_channel(P) endfunc

    function Istp_D_begin_dialogue_rc (P: PDU) : bool is
      Istp_begin_dialogue_rc(P) and is_dialogue(P) endfunc

    function Istp_C_begin_dialogue_rc (P: PDU) : bool is
      Istp_begin_dialogue_rc(P) and is_channel(P) endfunc

    function Istp_begin_dialogue_rc_accepted (P: PDU) : bool is
      Istp_begin_dialogue_rc(P) and is_accepted(get_bdr(P)) endfunc

    function Istp_begin_dialogue_rc_rejected (P: PDU) : bool is
      Istp_begin_dialogue_rc(P) and is_rejected(get_bdr(P)) endfunc

    function is_handshake_ri (P: PDU) : bool is
      Istp_handshake_ri(P) or Istp_handshake_and_grant_control_ri(P) endfunc

    function is_handshake_rc (P: PDU) : bool is
      Istp_handshake_rc(P) or Istp_handshake_and_grant_control_rc(P) endfunc

    function has_cnfu (P: PDU) : bool is
      (Istp_handshake_ri(P) and not(is_absent(get_cnfu0(P))))
      or
      Istp_handshake_and_grant_control_ri(P) endfunc

    function is_user (P: PDU) : bool is get_type (P) match user endfunc

    function is_provider (P: PDU) is get_type (P) match provider endfunc

/* equation shifted in module TP_PDU */
endmod
========================================================================== *)
```

```
type PDU is UPDU
opns
  Istp_D_begin_dialogue_ri,
  Istp_C_begin_dialogue_ri,
  Istp_D_begin_dialogue_rc,
  Istp_C_begin_dialogue_rc,
  Istp_begin_dialogue_rc_accepted,
  Istp_begin_dialogue_rc_rejected,
  is_handshake_ri,
  is_handshake_rc,
  has_cnfu :
    PDU -> bool
  is_user, is_provider : PDU -> bool
eqns
    forall pdu : PDU
  ofsort bool
    Istp_D_begin_dialogue_ri(pdu) =
      Istp_begin_dialogue_ri(pdu) and is_dialogue(pdu);
    Istp_C_begin_dialogue_ri(pdu) =
      Istp_begin_dialogue_ri(pdu) and is_channel(pdu);
    Istp_D_begin_dialogue_rc(pdu) =
      Istp_begin_dialogue_rc(pdu) and is_dialogue(pdu);
    Istp_C_begin_dialogue_rc(pdu) =
      Istp_begin_dialogue_rc(pdu) and is_channel(pdu);
    Istp_begin_dialogue_rc_accepted(pdu) =
      Istp_begin_dialogue_rc(pdu) and is_accepted(get_bdr(pdu));
    Istp_begin_dialogue_rc_rejected(pdu) =
      Istp_begin_dialogue_rc(pdu) and is_rejected(get_bdr(pdu));
    is_handshake_ri(pdu) =
      Istp_handshake_ri(pdu) or Istp_handshake_and_grant_control_ri(pdu);
    is_handshake_rc(pdu) =
      Istp_handshake_rc(pdu) or Istp_handshake_and_grant_control_rc(pdu);
    has_cnfu(pdu) =
      (Istp_handshake_ri(pdu) and not(is_absent(get_cnfu0(pdu))))
      or
      Istp_handshake_and_grant_control_ri(pdu);
    get_type(pdu) = user => is_user(pdu) = true;
    get_type(pdu) = provider => is_user(pdu) = false;
    is_provider(pdu) = not(is_user(pdu));
  ofsort confirmation_urgency
    Istp_handshake_ri(pdu) and not(is_absent(get_cnfu0(pdu))) =>
      get_cnfu(pdu) = get_cnfu(get_cnfu0(pdu));
endtype

(*
 *      H.5.4   Service Primitives
 *)
type BasicServicePrim is Boolean
sorts ServicePrim, ServicePrimKey
opns
  key : ServicePrim -> ServicePrimKey
  succ (*! constructor *) : ServicePrimKey -> ServicePrimKey
  _eq_, _ne_, _le_ : ServicePrimKey, ServicePrimKey -> bool
  _eq_ : ServicePrim, ServicePrimKey -> bool
endtype

type ServicePrim_equations is ServicePrim
eqns
    forall x, y : ServicePrimKey, sp : ServicePrim
  ofsort bool
    first_TPsp eq first_TPsp = true;
    succ(x) eq first_TPsp = false;
    first_TPsp eq succ(y) = false;
    succ(x) eq succ(y) = x eq y;
    x ne y = not(x eq y);
    first_TPsp le x = true;
    succ(x) le first_TPsp = false;
    succ(x) le succ(y) = x le y;
    sp eq x = key(sp) eq x;
endtype
(* -----------------------------------------------------------------------
module BasicPDU is Boolean
    type PDU is
    endtype

    type PDUkey is
        first_TPpdu /* defined in type TP_PDU */
    |   succ (K: PDUkey)
    endtype
/* no translation is needed for _eq_ */
    function _ne_ (P1: PDUkey, P2: PDUkey) : bool is not (P1 eq P2) endfunc

    function _le_ (P1: PDUkey, P2: PDUkey) : bool
        (P1 match first_TPpdu) or not(P2 match first_TPpdu) or
        ((select K in P1) le (select K in P2))
    endfunc

/* function key is specified in TP_PDU module */
/* function key (P: PDU) : PDUkey is endfunc */

    function _eq_ (P: PDU, K: PDUkey) is key(P) eq K endfunc
endmod
module PDU_equations is PDU
endmod
======================================================================= *)

type TP_ServicePrim is parameters, BasicServicePrim
opns
  TPDataReq (*! constructor *) : -> ServicePrim
  TPDataInd (*! constructor *) : -> ServicePrim
  TPBeginDialogueReq (*! constructor *) :
        (*initiating*) TPSU_title_Opt,
        (*recipient*) AP_title,
        (*recipient*) API_id_Opt,
        (*recipient*) AE_qual_Opt,
        (*recipient*) AEI_id_Opt,
        (*recipient*) TPSU_title_Opt,
        functional_units,
        quality_of_service_Opt,
        application_context_name,
        bool_Opt (*begin_transaction_Opt*),
        tp_begin_dialogue_confirmation,
        PDUqueue (*user_data*)  -> ServicePrim
  TPBeginDialogueInd (*! constructor *) :
        (*initiating*) AP_title_Opt,
        (*initiating*) API_id_Opt,
        (*initiating*) AE_qual_Opt,
        (*initiating*) AEI_id_Opt,
        (*initiating*) TPSU_title_Opt,
        functional_units,
        bool_Opt (*begin_transaction_Opt*),
        tp_begin_dialogue_confirmation,
        PDUqueue (*user_data*)  -> ServicePrim
  TPBeginDialogueRsp (*! constructor *) :
    tp_begin_dialogue_result, PDUqueue (*user_data*)  -> ServicePrim
  TPBeginDialogueCnf (*! constructor *) :
    functional_units_Opt,
    tp_begin_dialogue_result,
    tp_begin_dialogue_diagnostic_Opt,
    PDUqueue (*user_data*),
    bool (*rollback*)   -> ServicePrim
  TPEndDialogueReq (*! constructor *) : bool (*tp_end_dialogue_confirmation*)   -> ServicePrim
  TPEndDialogueInd (*! constructor *) : bool (*tp_end_dialogue_confirmation*)   -> ServicePrim
  TPEndDialogueRsp (*! constructor *) : -> ServicePrim
  TPEndDialogueCnf (*! constructor *) : -> ServicePrim
  TPUErrorReq (*! constructor *) : -> ServicePrim
  TPUErrorInd (*! constructor *) : -> ServicePrim
  TPUAbortReq (*! constructor *) : PDUqueue (*user_data*)  -> ServicePrim
  TPUAbortInd (*! constructor *) : PDUqueue (*user_data*), bool (*rollback*)    -> ServicePrim
  TPPAbortInd (*! constructor *) : tp_P_abort_diagnostic, bool (*rollback*)     -> ServicePrim
  TPGrantControlReq (*! constructor *) : -> ServicePrim
  TPGrantControlInd (*! constructor *) : -> ServicePrim
  TPRequestControlReq (*! constructor *) : -> ServicePrim
  TPRequestControlInd (*! constructor *) : -> ServicePrim
  TPHandshakeReq (*! constructor *) : confirmation_urgency_Opt -> ServicePrim
  TPHandshakeInd (*! constructor *) : -> ServicePrim
  TPHandshakeRsp (*! constructor *) : -> ServicePrim
  TPHandshakeCnf (*! constructor *) : -> ServicePrim
  TPHandshakeAndGrantControlReq (*! constructor *) : confirmation_urgency -> ServicePrim
  TPHandshakeAndGrantControlInd (*! constructor *) : -> ServicePrim
  TPHandshakeAndGrantControlRsp (*! constructor *) : -> ServicePrim
  TPHandshakeAndGrantControlCnf (*! constructor *) : -> ServicePrim
  TPHeuristicReportInd (*! constructor *) : heuristic_report -> ServicePrim
  TPBeginTransactionReq (*! constructor *) : -> ServicePrim
  TPBeginTransactionInd (*! constructor *) : -> ServicePrim
  TPDeferredEndDialogueReq (*! constructor *) : -> ServicePrim
  TPDeferredEndDialogueInd (*! constructor *) : -> ServicePrim
  TPDeferredGrantControlReq (*! constructor *) : -> ServicePrim
  TPDeferredGrantControlInd (*! constructor *) : -> ServicePrim
  TPCommitReq (*! constructor *) : -> ServicePrim
  TPCommitInd (*! constructor *) : -> ServicePrim
```

```
TPDoneReq (*! constructor *) : heuristic_report_Opt -> ServicePrim
TPCommitCompleteInd (*! constructor *) : -> ServicePrim
TPPrepareReq (*! constructor *) : bool_Opt (*data_permitted_Opt*) -> ServicePrim
TPPrepareInd (*! constructor *) : bool_Opt (*data_permitted_Opt*) -> ServicePrim
TPReadyInd (*! constructor *) : -> ServicePrim
TPRollbackReq (*! constructor *) : -> ServicePrim
TPRollbackInd (*! constructor *) : -> ServicePrim
TPRollbackCompleteInd (*! constructor *) : -> ServicePrim

TPData_req,
TPData_ind,
TPBeginDialogue_req,
TPBeginDialogue_ind,
TPBeginDialogue_rsp,
TPBeginDialogue_cnf,
TPEndDialogue_req,
TPEndDialogue_ind,
TPEndDialogue_rsp,
TPEndDialogue_cnf,
TPUError_req,
TPUError_ind,
TPUAbort_req,
TPUAbort_ind,
TPPAbort_ind,
TPGrantControl_req,
TPGrantControl_ind,
TPRequestControl_req,
TPRequestControl_ind,
TPHandshake_req,
TPHandshake_ind,
TPHandshake_rsp,
TPHandshake_cnf,
TPHandshakeAndGrantControl_req,
TPHandshakeAndGrantControl_ind,
TPHandshakeAndGrantControl_rsp,
TPHandshakeAndGrantControl_cnf,
TPHeuristicReport_ind,
TPBeginTransaction_req,
TPBeginTransaction_ind,
TPDeferredEndDialogue_req,
TPDeferredEndDialogue_ind,
TPDeferredGrantControl_req,
TPDeferredGrantControl_ind,
TPCommit_req,
TPCommit_ind,
TPDone_req,
TPCommitComplete_ind,
TPPrepare_req,
TPPrepare_ind,
TPReady_ind,
TPRollback_req,
TPRollback_ind,
TPRollbackComplete_ind,
first_TPsp (*! constructor *),
last_TPsp : ->
    ServicePrimKey
IsTPDataReq,
IsTPDataInd,
IsTPBeginDialogueReq,
IsTPBeginDialogueInd,
IsTPBeginDialogueRsp,
IsTPBeginDialogueCnf,
IsTPEndDialogueReq,
IsTPEndDialogueInd,
IsTPEndDialogueRsp,
IsTPEndDialogueCnf,
IsTPUErrorReq,
IsTPUErrorInd,
IsTPUAbortReq,
IsTPUAbortInd,
IsTPPAbortInd,
IsTPGrantControlReq,
IsTPGrantControlInd,
IsTPRequestControlReq,
IsTPRequestControlInd,
IsTPHandshakeReq,
IsTPHandshakeInd,
IsTPHandshakeRsp,
IsTPHandshakeCnf,
IsTPHandshakeAndGrantControlReq,
IsTPHandshakeAndGrantControlInd,
IsTPHandshakeAndGrantControlRsp,
IsTPHandshakeAndGrantControlCnf,
IsTPHeuristicReportInd,
IsTPBeginTransactionReq,
IsTPBeginTransactionInd,
IsTPDeferredEndDialogueReq,
IsTPDeferredEndDialogueInd,
IsTPDeferredGrantControlReq,
IsTPDeferredGrantControlInd,
IsTPCommitReq,
IsTPCommitInd,
IsTPDoneReq,
IsTPCommitCompleteInd,
IsTPPrepareReq,
IsTPPrepareInd,
IsTPReadyInd,
IsTPRollbackReq,
IsTPRollbackInd,
IsTPRollbackCompleteInd,
IsTPsp :
  ServicePrim -> bool
get_acn : ServicePrim -> application_context_name
get_bdc : ServicePrim -> tp_begin_dialogue_confirmation
get_bddO : ServicePrim -> tp_begin_dialogue_diagnostic_Opt
get_bdr : ServicePrim -> tp_begin_dialogue_result
get_btO : ServicePrim -> bool_Opt (*begin_transaction_Opt*)
get_cnfu :
    ServicePrim -> confirmation_urgency
  get_cnfuO : ServicePrim -> confirmation_urgency_Opt
  get_dpO : ServicePrim -> bool_Opt (*data_permitted_Opt*)
  get_edc :
    ServicePrim -> bool (*tp_end_dialogue_confirmation*)
  get_hr :
    ServicePrim -> heuristic_report
  get_hrO : ServicePrim -> heuristic_report_Opt
  get_iaeiO : ServicePrim -> (*initiating*) AEI_id_Opt
  get_iaeqO : ServicePrim -> (*initiating*) AE_qual_Opt
  get_iapiO : ServicePrim -> (*initiating*) API_id_Opt
  get_iaptO : ServicePrim -> (*initiating*) AP_title_Opt
  get_ittO : ServicePrim -> (*initiating*) TPSU_title_Opt
  get_pad : ServicePrim -> tp_P_abort_diagnostic
  get_qosO : ServicePrim -> quality_of_service_Opt
  get_raeiO : ServicePrim -> (*recipient*) AEI_id_Opt
  get_raeqO : ServicePrim -> (*recipient*) AE_qual_Opt
  get_rapiO : ServicePrim -> (*recipient*) API_id_Opt
  get_rapt : ServicePrim -> (*recipient*) AP_title
  get_rbk : ServicePrim -> bool (*rollback*)
  get_rttO :
    ServicePrim -> (*recipient*) TPSU_title_Opt
  get_sfu : ServicePrim -> functional_units
  get_fuO : ServicePrim -> functional_units_Opt
  get_ud : ServicePrim -> PDUqueue (*user_data*)
eqns
    forall
      acn :
      application_context_name,
      bdc : tp_begin_dialogue_confirmation,
      bddO : tp_begin_dialogue_diagnostic_Opt,
      bdr : tp_begin_dialogue_result,
      btO : bool_Opt (*begin_transaction_Opt*),
      cnfu : confirmation_urgency,
      cnfuO : confirmation_urgency_Opt,
      dpO : bool_Opt (*data_permitted_Opt*),
      edc : bool (*tp_end_dialogue_confirmation*),
      hr : heuristic_report,
      hrO : heuristic_report_Opt,
      iaeiO : (*initiating*) AEI_id_Opt,
      iaeqO : (*initiating*) AE_qual_Opt,
      iapiO : (*initiating*) API_id_Opt,
      iaptO : (*initiating*) AP_title_Opt,
      ittO :  (*initiating*) TPSU_title_Opt,
      pad : tp_P_abort_diagnostic,
      qosO : quality_of_service_Opt,
      raeiO : (*recipient*) AEI_id_Opt,
      raeqO : (*recipient*) AE_qual_Opt,
      rapiO : (*recipient*) API_id_Opt,
      rapt :  (*recipient*) AP_title,
      rbk : bool (*rollback*),
      rttO :  (*recipient*) TPSU_title_Opt,
      sfu : functional_units,
      fuO : functional_units_Opt,
      ud : PDUqueue (*user_data*),
      sp : ServicePrim
  ofsort ServicePrimKey
    TPData_req = first_TPsp;
    TPData_ind = succ(TPData_req);
    TPBeginDialogue_req = succ(TPData_ind);
    TPBeginDialogue_ind = succ(TPBeginDialogue_req);
    TPBeginDialogue_rsp = succ(TPBeginDialogue_ind);
    TPBeginDialogue_cnf = succ(TPBeginDialogue_rsp);
    TPEndDialogue_req = succ(TPBeginDialogue_cnf);
    TPEndDialogue_ind = succ(TPEndDialogue_req);
    TPEndDialogue_rsp = succ(TPEndDialogue_ind);
    TPEndDialogue_cnf = succ(TPEndDialogue_rsp);
    TPUError_req = succ(TPEndDialogue_cnf);
    TPUError_ind = succ(TPUError_req);
    TPUAbort_req = succ(TPUError_ind);
    TPUAbort_ind = succ(TPUAbort_req);
    TPPAbort_ind = succ(TPUAbort_ind);
    TPGrantControl_req = succ(TPPAbort_ind);
    TPGrantControl_ind = succ(TPGrantControl_req);
    TPRequestControl_req = succ(TPGrantControl_ind);
    TPRequestControl_ind = succ(TPRequestControl_req);
    TPHandshake_req = succ(TPRequestControl_ind);
    TPHandshake_ind = succ(TPHandshake_req);
    TPHandshake_rsp = succ(TPHandshake_ind);
    TPHandshake_cnf = succ(TPHandshake_rsp);
    TPHandshakeAndGrantControl_req = succ(TPHandshake_cnf);
    TPHandshakeAndGrantControl_ind = succ(TPHandshakeAndGrantControl_req);
    TPHandshakeAndGrantControl_rsp = succ(TPHandshakeAndGrantControl_ind);
    TPHandshakeAndGrantControl_cnf = succ(TPHandshakeAndGrantControl_rsp);
    TPHeuristicReport_ind = succ(TPHandshakeAndGrantControl_cnf);
    TPBeginTransaction_req = succ(TPHeuristicReport_ind);
    TPBeginTransaction_ind = succ(TPBeginTransaction_req);
    TPDeferredEndDialogue_req = succ(TPBeginTransaction_ind);
    TPDeferredEndDialogue_ind = succ(TPDeferredEndDialogue_req);
    TPDeferredGrantControl_req = succ(TPDeferredEndDialogue_ind);
    TPDeferredGrantControl_ind = succ(TPDeferredGrantControl_req);
    TPCommit_req = succ(TPDeferredGrantControl_ind);
    TPCommit_ind = succ(TPCommit_req);
    TPDone_req = succ(TPCommit_ind);
    TPCommitComplete_ind = succ(TPDone_req);
    TPPrepare_req = succ(TPCommitComplete_ind);
    TPPrepare_ind = succ(TPPrepare_req);
    TPReady_ind = succ(TPPrepare_ind);
    TPRollback_req = succ(TPReady_ind);
    TPRollback_ind = succ(TPRollback_req);
    TPRollbackComplete_ind = succ(TPRollback_ind);
    last_TPsp = TPRollbackComplete_ind;
    key(TPDataReq) = TPData_req;
    key(TPDataInd) = TPData_ind;
    key(
      TPBeginDialogueReq(
        itt0, rapt, rapi0, raeq0, raei0, rtt0, sfu, qos0, acn, bt0, bdc, ud)) =
      TPBeginDialogue_req;
    key(
      TPBeginDialogueInd(iapt0, iapi0, iaeq0, iaei0, itt0, sfu, bt0, bdc, ud)) =
      TPBeginDialogue_ind;
    key(TPBeginDialogueRsp(bdr, ud)) = TPBeginDialogue_rsp;
    key(TPBeginDialogueCnf(fu0, bdr, bdd0, ud, rbk)) = TPBeginDialogue_cnf;
    key(TPEndDialogueReq(edc)) = TPEndDialogue_req;
    key(TPEndDialogueInd(edc)) = TPEndDialogue_ind;
    key(TPEndDialogueRsp) = TPEndDialogue_rsp;
    key(TPEndDialogueCnf) = TPEndDialogue_cnf;
    key(TPUErrorReq) = TPUError_req;
    key(TPUErrorInd) = TPUError_ind;
    key(TPUAbortReq(ud)) = TPUAbort_req;
    key(TPUAbortInd(ud, rbk)) = TPUAbort_ind;
    key(TPPAbortInd(pad, rbk)) = TPPAbort_ind;
    key(TPGrantControlReq) = TPGrantControl_req;
    key(TPGrantControlInd) = TPGrantControl_ind;
    key(TPRequestControlReq) = TPRequestControl_req;
    key(TPRequestControlInd) = TPRequestControl_ind;
    key(TPHandshakeReq(cnfu0)) = TPHandshake_req;
    key(TPHandshakeInd) = TPHandshake_ind;
    key(TPHandshakeRsp) = TPHandshake_rsp;
    key(TPHandshakeCnf) = TPHandshake_cnf;
    key(TPHandshakeAndGrantControlReq(cnfu)) = TPHandshakeAndGrantControl_req;
    key(TPHandshakeAndGrantControlInd) = TPHandshakeAndGrantControl_ind;
    key(TPHandshakeAndGrantControlRsp) = TPHandshakeAndGrantControl_rsp;
    key(TPHandshakeAndGrantControlCnf) = TPHandshakeAndGrantControl_cnf;
    key(TPHeuristicReportInd(hr)) = TPHeuristicReport_ind;
    key(TPBeginTransactionReq) = TPBeginTransaction_req;
    key(TPBeginTransactionInd) = TPBeginTransaction_ind;
    key(TPDeferredEndDialogueReq) = TPDeferredEndDialogue_req;
    key(TPDeferredEndDialogueInd) = TPDeferredEndDialogue_ind;
    key(TPDeferredGrantControlReq) = TPDeferredGrantControl_req;
    key(TPDeferredGrantControlInd) = TPDeferredGrantControl_ind;
    key(TPCommitReq) = TPCommit_req;
    key(TPCommitInd) = TPCommit_ind;
    key(TPDoneReq(hr0)) = TPDone_req;
    key(TPCommitCompleteInd) = TPCommitComplete_ind;
    key(TPPrepareReq(dp0)) = TPPrepare_req;
    key(TPPrepareInd(dp0)) = TPPrepare_ind;
    key(TPReadyInd) = TPReady_ind;
    key(TPRollbackReq) = TPRollback_req;
    key(TPRollbackInd) = TPRollback_ind;
    key(TPRollbackCompleteInd) = TPRollbackComplete_ind;
  ofsort bool
    IsTPDataReq(sp) = sp eq TPData_req;
    IsTPDataInd(sp) = sp eq TPData_ind;
    IsTPBeginDialogueReq(sp) = sp eq TPBeginDialogue_req;
    IsTPBeginDialogueInd(sp) = sp eq TPBeginDialogue_ind;
    IsTPBeginDialogueRsp(sp) = sp eq TPBeginDialogue_rsp;
    IsTPBeginDialogueCnf(sp) = sp eq TPBeginDialogue_cnf;
    IsTPEndDialogueReq(sp) = sp eq TPEndDialogue_req;
    IsTPEndDialogueInd(sp) = sp eq TPEndDialogue_ind;
    IsTPEndDialogueRsp(sp) = sp eq TPEndDialogue_rsp;
    IsTPEndDialogueCnf(sp) = sp eq TPEndDialogue_cnf;
    IsTPUErrorReq(sp) = sp eq TPUError_req;
    IsTPUErrorInd(sp) = sp eq TPUError_ind;
    IsTPUAbortReq(sp) = sp eq TPUAbort_req;
    IsTPUAbortInd(sp) = sp eq TPUAbort_ind;
    IsTPPAbortInd(sp) = sp eq TPPAbort_ind;
    IsTPGrantControlReq(sp) = sp eq TPGrantControl_req;
    IsTPGrantControlInd(sp) = sp eq TPGrantControl_ind;
    IsTPRequestControlReq(sp) = sp eq TPRequestControl_req;
    IsTPRequestControlInd(sp) = sp eq TPRequestControl_ind;
    IsTPHandshakeReq(sp) = sp eq TPHandshake_req;
    IsTPHandshakeInd(sp) = sp eq TPHandshake_ind;
    IsTPHandshakeRsp(sp) = sp eq TPHandshake_rsp;
    IsTPHandshakeCnf(sp) = sp eq TPHandshake_cnf;
    IsTPHandshakeAndGrantControlReq(sp) = sp eq TPHandshakeAndGrantControl_req;
    IsTPHandshakeAndGrantControlInd(sp) = sp eq TPHandshakeAndGrantControl_ind;
    IsTPHandshakeAndGrantControlRsp(sp) = sp eq TPHandshakeAndGrantControl_rsp;
    IsTPHandshakeAndGrantControlCnf(sp) = sp eq TPHandshakeAndGrantControl_cnf;
    IsTPHeuristicReportInd(sp) = sp eq TPHeuristicReport_ind;
    IsTPBeginTransactionReq(sp) = sp eq TPBeginTransaction_req;
    IsTPBeginTransactionInd(sp) = sp eq TPBeginTransaction_ind;
    IsTPDeferredEndDialogueReq(sp) = sp eq TPDeferredEndDialogue_req;
    IsTPDeferredEndDialogueInd(sp) = sp eq TPDeferredEndDialogue_ind;
    IsTPDeferredGrantControlReq(sp) = sp eq TPDeferredGrantControl_req;
    IsTPDeferredGrantControlInd(sp) = sp eq TPDeferredGrantControl_ind;
    IsTPCommitReq(sp) = sp eq TPCommit_req;
    IsTPCommitInd(sp) = sp eq TPCommit_ind;
    IsTPDoneReq(sp) = sp eq TPDone_req;
    IsTPCommitCompleteInd(sp) = sp eq TPCommitComplete_ind;
    IsTPPrepareReq(sp) = sp eq TPPrepare_req;
    IsTPPrepareInd(sp) = sp eq TPPrepare_ind;
    IsTPReadyInd(sp) = sp eq TPReady_ind;
    IsTPRollbackReq(sp) = sp eq TPRollback_req;
    IsTPRollbackInd(sp) = sp eq TPRollback_ind;
    IsTPRollbackCompleteInd(sp) = sp eq TPRollbackComplete_ind;
    IsTPsp(sp) = (first_TPsp le key(sp)) and (key(sp) le last_TPsp);
  ofsort application_context_name
    get_acn(
      TPBeginDialogueReq(
        itt0, rapt, rapi0, raeq0, raei0, rtt0, sfu, qos0, acn, bt0, bdc, ud)) =
      acn;
  ofsort tp_begin_dialogue_confirmation
    get_bdc(
      TPBeginDialogueReq(
        itt0, rapt, rapi0, raeq0, raei0, rtt0, sfu, qos0, acn, bt0, bdc, ud)) =
      bdc;
    get_bdc(
      TPBeginDialogueInd(iapt0, iapi0, iaeq0, iaei0, itt0, sfu, bt0, bdc, ud)) =
      bdc;
  ofsort tp_begin_dialogue_diagnostic_Opt
    get_bdd0(TPBeginDialogueCnf(fu0, bdr, bdd0, ud, rbk)) = bdd0;
  ofsort tp_begin_dialogue_result
    get_bdr(TPBeginDialogueRsp(bdr, ud)) = bdr;
    get_bdr(TPBeginDialogueCnf(fu0, bdr, bdd0, ud, rbk)) = bdr;
  ofsort bool_Opt (*begin_transaction_Opt*)
```

```
    get_btO(
      TPBeginDialogueReq(
        ittO, rapt, rapiO, raeqO, raeiO, rttO, sfu, qosO, acn, btO, bdc, ud)) =
      btO;
    get_btO(
      TPBeginDialogueInd(iaptO, iapiO, iaeqO, iaeiO, ittO, sfu, btO, bdc, ud)) =
      btO;
  ofsort confirmation_urgency
    get_cnfu(TPHandshakeAndGrantControlReq(cnfu)) = cnfu;
  ofsort confirmation_urgency_Opt get_cnfuO(TPHandshakeReq(cnfuO)) = cnfuO;
  ofsort bool_Opt (*data_permitted_Opt*)
    get_dpO(
      TPPrepareReq(dpO)) =
      dpO;
    get_dpO(TPPrepareInd(dpO)) = dpO;
  ofsort bool (*tp_end_dialogue_confirmation*)
    get_edc(
      TPEndDialogueReq(edc)) =
      edc;
    get_edc(TPEndDialogueInd(edc)) = edc;
  ofsort heuristic_report
    get_hr(TPHeuristicReportInd(hr)) = hr; get_hr(TPDoneReq(Opt(hr))) = hr;
  ofsort heuristic_report_Opt get_hrO(TPDoneReq(hrO)) = hrO;
  ofsort  (*initiating*) AEI_id_Opt
    get_iaeiO(
      TPBeginDialogueInd(iaptO, iapiO, iaeqO, iaeiO, ittO, sfu, btO, bdc, ud)) =
      iaeiO;
  ofsort  (*initiating*) AE_qual_Opt
    get_iaeqO(
      TPBeginDialogueInd(iaptO, iapiO, iaeqO, iaeiO, ittO, sfu, btO, bdc, ud)) =
      iaeqO;
  ofsort  (*initiating*) API_id_Opt
    get_iapiO(
      TPBeginDialogueInd(iaptO, iapiO, iaeqO, iaeiO, ittO, sfu, btO, bdc, ud)) =
      iapiO;
  ofsort  (*initiating*) AP_title_Opt
    get_iaptO(
      TPBeginDialogueInd(iaptO, iapiO, iaeqO, iaeiO, ittO, sfu, btO, bdc, ud)) =
      iaptO;
  ofsort  (*initiating*) TPSU_title_Opt
    get_ittO(
      TPBeginDialogueReq(
        ittO, rapt, rapiO, raeqO, raeiO, rttO, sfu, qosO, acn, btO, bdc, ud)) =
      ittO;
    get_ittO(
      TPBeginDialogueInd(iaptO, iapiO, iaeqO, iaeiO, ittO, sfu, btO, bdc, ud)) =
      ittO;
  ofsort tp_P_abort_diagnostic get_pad(TPPAbortInd(pad, rbk)) = pad;
  ofsort quality_of_service_Opt
    get_qosO(
      TPBeginDialogueReq(
        ittO, rapt, rapiO, raeqO, raeiO, rttO, sfu, qosO, acn, btO, bdc, ud)) =
      qosO;
  ofsort  (*recipient*) AEI_id_Opt
```

```
    get_raeiO(
      TPBeginDialogueReq(
        ittO, rapt, rapiO, raeqO, raeiO, rttO, sfu, qosO, acn, btO, bdc, ud)) =
      raeiO;
  ofsort  (*recipient*) AE_qual_Opt
    get_raeqO(
      TPBeginDialogueReq(
        ittO, rapt, rapiO, raeqO, raeiO, rttO, sfu, qosO, acn, btO, bdc, ud)) =
      raeqO;
  ofsort  (*recipient*) API_id_Opt
    get_rapiO(
      TPBeginDialogueReq(
        ittO, rapt, rapiO, raeqO, raeiO, rttO, sfu, qosO, acn, btO, bdc, ud)) =
      rapiO;
  ofsort  (*recipient*) AP_title
    get_rapt(
      TPBeginDialogueReq(
        ittO, rapt, rapiO, raeqO, raeiO, rttO, sfu, qosO, acn, btO, bdc, ud)) =
      rapt;
  ofsort bool (*rollback*)
    get_rbk(
      TPBeginDialogueCnf(fuO, bdr, bddO, ud, rbk)) =
      rbk;
    get_rbk(TPUAbortInd(ud, rbk)) = rbk;
    get_rbk(TPPAbortInd(pad, rbk)) = rbk;
  ofsort  (*recipient*) TPSU_title_Opt
    get_rttO(
      TPBeginDialogueReq(
        ittO, rapt, rapiO, raeqO, raeiO, rttO, sfu, qosO, acn, btO, bdc, ud)) =
      rttO;
  ofsort functional_units
    get_sfu(
      TPBeginDialogueReq(
        ittO, rapt, rapiO, raeqO, raeiO, rttO, sfu, qosO, acn, btO, bdc, ud)) =
      sfu;
    get_sfu(
      TPBeginDialogueInd(iaptO, iapiO, iaeqO, iaeiO, ittO, sfu, btO, bdc, ud)) =
      sfu;
  ofsort functional_units_Opt
    get_fuO(TPBeginDialogueCnf(fuO, bdr, bddO, ud, rbk)) = fuO;
  ofsort PDUqueue (*user_data*)
    get_ud(
      TPBeginDialogueReq(
        ittO, rapt, rapiO, raeqO, raeiO, rttO, sfu, qosO, acn, btO, bdc, ud)) =
      ud;
    get_ud(
      TPBeginDialogueInd(iaptO, iapiO, iaeqO, iaeiO, ittO, sfu, btO, bdc, ud)) =
      ud;
    get_ud(TPBeginDialogueRsp(bdr, ud)) = ud;
    get_ud(TPBeginDialogueCnf(fuO, bdr, bddO, ud, rbk)) = ud;
    get_ud(TPUAbortReq(ud)) = ud;
    get_ud(TPUAbortInd(ud, rbk)) = ud;
endtype
(* -------------------------------------------------------------------------
module TP_ServicePrim is parameters, BasicServicePrim

    type TP_ServicePrim is
        TPDataReq
    |   TPDataInd
    |   TPBeginDialogueReq (ITTO: TPSU_title_Opt, /* initiating */
                            RAPT: AP_title,     /* recipient */
                            RAPIO: API_id_Opt,  /*recipient*/
                            RAEQO: AE_qual_Opt, /*recipient*/
                            RAEIO: AEI_id_Opt,  /*recipient*/
                            RTTO: TPSU_title_Opt,
                            SFU: functional_units,
                            QOSO: quality_of_service_Opt,
                            ACN: application_context_name,
                            BTO: bool_Opt       /*begin_transaction_Opt*/,
                            BDC: tp_begin_dialogue_confirmation,
                            UD: PDUqueue)       /*user_data*/
    |   TPBeginDialogueInd (IAPTO: AP_title_Opt,/* initiating */
                            IAPIO: API_id_Opt,  /* initiating */
                            IAEQO: AE_qual_Opt, /* initiating */
                            IAEIO: AEI_id_Opt,  /* initiating */
                            ITTO: TPSU_title_Opt,
                            SFU: functional_units,
                            BTO: bool_Opt,      /*begin_transaction_Opt*/
                            BDC: tp_begin_dialogue_confirmation,
                            UD: PDUqueue)       /*user_data*/
    |   TPBeginDialogueRsp (BDR: tp_begin_dialogue_result,
                            UD: PDUqueue)       /*user_data*/
    |   TPBeginDialogueCnf (FUO: functional_units_Opt,
                            BDR: tp_begin_dialogue_result,
                            BDDO: tp_begin_dialogue_diagnostic_Opt,
                            UD: PDUqueue,       /*user_data*/
                            rbk: bool)          /*rollback*/
    |   TPEndDialogueReq   (EDC: bool)  /*tp_end_dialogue_confirmation*/
    |   TPEndDialogueInd   (EDC: bool)  /*tp_end_dialogue_confirmation*/
    |   TPEndDialogueRsp
    |   TPEndDialogueCnf
    |   TPUErrorReq
    |   TPUErrorInd
    |   TPUAbortReq        (UD: PDUqueue)       /*user_data*/
    |   TPUAbortInd        (UD: PDUqueue, RBK: bool)
    |   TPPAbortInd        (PAD: tp_P_abort_diagnostic, RBK: bool)
    |   TPGrantControlReq
    |   TPGrantControlInd
    |   TPRequestControlReq
    |   TPRequestControlInd
    |   TPHandshakeReq     (CNFUO: confirmation_urgency_Opt)
    |   TPHandshakeInd
    |   TPHandshakeRsp
    |   TPHandshakeCnf
    |   TPHandshakeAndGrantControlReq (CNFU: confirmation_urgency)
    |   TPHandshakeAndGrantControlInd
    |   TPHandshakeAndGrantControlRsp
    |   TPHandshakeAndGrantControlCnf
    |   TPHeuristicReportInd (HR: heuristic_report)
    |   TPBeginTransactionReq
    |   TPBeginTransactionInd
    |   TPDeferredEndDialogueReq
    |   TPDeferredEndDialogueInd
    |   TPDeferredGrantControlReq
    |   TPDeferredGrantControlInd
    |   TPCommitReq
    |   TPCommitInd
    |   TPDoneReq           (HRO: heuristic_report_Opt)
    |   TPCommitCompleteInd
    |   TPPrepareReq        (DPO: bool_Opt)     /*data_permitted_Opt*/
    |   TPPrepareInd        (DPO: bool_Opt)     /*data_permitted_Opt*/
    |   TPReadyInd
    |   TPRollbackReq
    |   TPRollbackInd
    |   TPRollbackCompleteInd
    endtype

/* no translation is needed for TP*_{req|ind|cnf} */
/* no translation is needed for function key */

    function IsTPDataReq (SP: TP_ServicePrim) : bool is SP match TPDataReq endfunc
    function IsTPDataInd (SP: TP_ServicePrim) : bool is SP match TPDataInd endfunc
    function IsTPBeginDialogueReq (SP: TP_ServicePrim) : bool is SP match TPBeginDialogueReq (...) endfunc
    function IsTPBeginDialogueInd (SP: TP_ServicePrim) : bool is SP match TPBeginDialogueInd (...) endfunc
    function IsTPBeginDialogueRSP (SP: TP_ServicePrim) : bool is SP match TPBeginDialogueRsp (...) endfunc
    function IsTPBeginDialogueCnf (SP: TP_ServicePrim) : bool is SP match TPBeginDialogueCnf endfunc
    function IsTPEndDialogueReq (SP: TP_ServicePrim) : bool is SP match TPEndDialogueReq (...) endfunc
    function IsTPEndDialogueInd (SP: TP_ServicePrim) : bool is SP match TPEndDialogueInd (...) endfunc
    function IsTPEndDialogueRsp (SP: TP_ServicePrim) : bool is SP match TPEndDialogueRsp endfunc
    function IsTPEndDialogueCnf (SP: TP_ServicePrim) : bool is SP match TPEndDialogueCnf endfunc
    function IsTPUErrorReq (SP: TP_ServicePrim) : bool is SP match TPUErrorReq endfunc
    function IsTPUErrorInd (SP: TP_ServicePrim) : bool is SP match TPUErrorInd endfunc
    function IsTPUAbortReq (SP: TP_ServicePrim) : bool is SP match TPUAbortReq (...) endfunc
    function IsTPUAbortInd (SP: TP_ServicePrim) : bool is SP match TPUAbortInd (...) endfunc
    function IsTPPAbortInd (SP: TP_ServicePrim) : bool is SP match TPPAbortInd (...) endfunc
    function IsTPGrantControlReq (SP: TP_ServicePrim) : bool is SP match TPGrantControlReq endfunc
    function IsTPGrantControlInd (SP: TP_ServicePrim) : bool is SP match TPGrantControlInd endfunc
    function IsTPRequestControlReq (SP: TP_ServicePrim) : bool is SP match TPRequestControlReq endfunc
    function IsTPRequestControlInd (SP: TP_ServicePrim) : bool is SP match TPRequestControlInd endfunc
    function IsTPHandshakeReq (SP: TP_ServicePrim) : bool is SP match TPHandshakeReq (...) endfunc
    function IsTPHandshakeInd (SP: TP_ServicePrim) : bool is SP match TPHandshakeInd endfunc
    function IsTPHandshakeRSP (SP: TP_ServicePrim) : bool is SP match TPHandshakeRsp endfunc
    function IsTPHandshakeCnf (SP: TP_ServicePrim) : bool is SP match TPHandshakeCnf endfunc
    function IsTPHandshakeAndGrantControlReq (SP: TP_ServicePrim) : bool is
        SP match TPHandshakeAndGrantControlReq (...) endfunc
    function IsTPHandshakeAndGrantControlInd (SP: TP_ServicePrim) : bool is
        SP match TPHandshakeAndGrantControlInd endfunc
    function IsTPHandshakeAndGrantControlRSP (SP: TP_ServicePrim) : bool is
        SP match TPHandshakeAndGrantControlRsp endfunc
    function IsTPHandshakeAndGrantControlCnf (SP: TP_ServicePrim) : bool is
        SP match TPHandshakeAndGrantControlCnf endfunc
    function IsTPHeuristicReportInd (SP: TP_ServicePrim) : bool is
```

```
        SP match TPHeuristicReportInd (...) endfunc
    function IsTPBeginTransactionReq (SP: TP_ServicePrim) : bool is
        SP match TPBeginTransactionReq endfunc
    function IsTPBeginTransactionInd (SP: TP_ServicePrim) : bool is
        SP match TPBeginTransactionInd endfunc
    function IsTPDeferredEndDialogueReq (SP: TP_ServicePrim) : bool is
        SP match TPDeferredEndDialogueReq endfunc
    function IsTPDeferredEndDialogueInd (SP: TP_ServicePrim) : bool is
        SP match TPDeferredEndDialogueInd endfunc
    function IsTPDeferredGrantControlReq (SP: TP_ServicePrim) : bool is
        SP match TPDeferredGrantControlReq endfunc
    function IsTPDeferredGrantControlInd (SP: TP_ServicePrim) : bool is
        SP match TPDeferredGrantControlInd endfunc
    function IsTPCommitReq (SP: TP_ServicePrim) : bool is SP match TPCommitReq endfunc
    function IsTPCommitInd (SP: TP_ServicePrim) : bool is SP match TPCommitInd endfunc
    function IsTPDoneReq (SP: TP_ServicePrim) : bool is SP match TPDoneReq (...) endfunc
    function IsTPCommitCompleteInd (SP: TP_ServicePrim) : bool is SP match TPCommitCompleteInd endfunc
    function IsTPPrepareReq (SP: TP_ServicePrim) : bool is SP match TPPrepareReq (...) endfunc
    function IsTPPrepareInd (SP: TP_ServicePrim) : bool is SP match TPPrepareInd (...) endfunc
    function IsTPReadyInd (SP: TP_ServicePrim) : bool is SP match TPReadyInd endfunc
    function IsTPRollbackReq (SP: TP_ServicePrim) : bool is SP match TPRollbackReq endfunc
    function IsTPRollbackInd (SP: TP_ServicePrim) : bool is SP match TPRollbackInd endfunc
    function IsTPRollbackCompleteInd (SP: TP_ServicePrim) : bool is SP match TPRollbackCompleteInd endfunc
    function IsTPsp (SP: ServicePrim) : bool is SP match TP (...) endfunc
/* (first_TPsp le key(SP)) and (key(SP) le last_TPsp) */

    function get_acn (SP: TP_ServicePrim) : application_context_name is select ACN in SP endfunc

    function get_bdc (SP: TP_ServicePrim) : tp_begin_dialogue_confirmation is select BDC in SP endfunc

    function get_bddO (SP: TP_ServicePrim) : tp_begin_dialogue_diagnostic_Opt is select BDDO in SP endfunc

    function get_bdr (SP: TP_ServicePrim) : tp_begin_dialogue_result is select BDR in SP endfunc

    function get_btO (SP: TP_ServicePrim) : bool_Opt is select BTO in SP endfunc

    function get_cnfu (SP: TP_ServicePrim) : confirmation_urgency is select CNFU in SP endfunc

    function get_cnfuO (SP: TP_ServicePrim) : confirmation_urgency_Opt is select CNFUO in SP endfunc

    function get_dpO (SP: TP_ServicePrim) : bool_Opt is select DPO in SP endfunc

    function get_edc (SP: TP_ServicePrim) : bool is select EDC in SP endfunc

    function get_hr (SP: TP_ServicePrim) : heuristic_report is select HR in SP endfunc

    function get_hrO (SP: TP_ServicePrim) : heuristic_report_Opt is select HRO in SP endfunc

    function get_iaeiO (SP: TP_ServicePrim) : AEI_id_Opt is select IAEIO in SP endfunc

    function get_iaeqO (SP: TP_ServicePrim) : AE_qual_Opt is select IAEQO in SP endfunc

    function get_iapiO (SP: TP_ServicePrim) : API_id_Opt is select IAPIO in SP endfunc
    function get_iaptO (SP: TP_ServicePrim) : AP_title_Opt is select IAPTO in SP endfunc

    function get_ittO (SP: TP_ServicePrim) : TPSU_title_Opt is select ITTO in SP endfunc

    function get_pad (SP: TP_ServicePrim) : tp_P_abort_diagnostic  is select PAD in SP endfunc

    function get_qosO (SP: TP_ServicePrim) : quality_of_service_Opt is select QOSO in SP endfunc

    function get_raeiO (SP: TP_ServicePrim) : AEI_id_Opt is select RAEIO in SP endfunc

    function get_raeqO (SP: TP_ServicePrim) : AE_qual_Opt is select RAEQO in SP endfunc

    function get_rapiO (SP: TP_ServicePrim) : API_id_Opt is select API_id_Opt in SP endfunc

    function get_rapt (SP: TP_ServicePrim) : AP_title is select RAPT in SP endfunc

    function get_rbk (SP: TP_ServicePrim) : bool is select RBK in SP endfunc

    function get_rttO (SP: TP_ServicePrim) : TPSU_title_Opt is select RTTO in SP endfunc

    function get_sfu (SP: TP_ServicePrim) : functional_units is select SFU in SP endfunc

    function get_fuO (SP: TP_ServicePrim) :  is select FUO in SP endfunc

    function get_ud (SP: TP_ServicePrim) : PDUqueue is elect UD in SP endfunc

endmod
=========================================================================== *)

type UASE_ServicePrim is TP_ServicePrim
opns
        (*  UASE SPs are identified by IsTPDataReq and IsTPDataInd  *)
  IsUASEReq, IsUASEInd, IsUASEsp : ServicePrim -> bool
eqns
    forall sp : ServicePrim
  ofsort bool
    IsUASEReq(sp) = IsTPDataReq(sp);
    IsUASEInd(sp) = IsTPDataInd(sp);
    IsUASEsp(sp) = IsUASEReq(sp) or IsUASEInd(sp);
endtype
(* ---------------------------------------------------------------------------
module UASE_ServicePrim is TP_ServicePrim

    function IsUASEReq (SP: TP_ServicePrim) : bool is IsTPDataReq(SP) endfunc

    function IsUASEInd (SP: TP_ServicePrim) : bool is IsTPDataInd(SP) endfunc

    function IsUASEsp (SP: TP_ServicePrim) : bool is IsUASEReq(SP) or IsUASEInd(SP) endfunc
endmod
=========================================================================== *)

type AF_ServicePrim is UASE_ServicePrim
opns
  AFBeginDialogueReq (*! constructor *) :
    (*initiating*) TPSU_title_Opt,
    (*recipient*) TPSU_title_Opt,
    functional_units,
    bool_Opt (*begin_transaction_Opt*),
    tp_begin_dialogue_confirmation,
    correlator_Opt,
    correlator_Opt (*last_partner_identifier*),
    PDUqueue (*user_data*)  -> ServicePrim
  AFBeginDialogueReq (*! constructor *) :
    functional_units,
    correlator_Opt,
    channel_utilization,
    correlator_Opt (*last_partner_identifier*)      -> ServicePrim
  AFBeginDialogueInd (*! constructor *) :
    (*initiating*) TPSU_title_Opt,
    (*recipient*) TPSU_title_Opt,
    functional_units,
    bool_Opt (*begin_transaction_Opt*),
    tp_begin_dialogue_confirmation,
    correlator_Opt,
    correlator_Opt (*last_partner_identifier*),
    PDUqueue (*user_data*)  -> ServicePrim
  AFBeginDialogueInd (*! constructor *) :
    functional_units,
    correlator_Opt,
    channel_utilization,
    correlator_Opt (*last_partner_identifier*)      -> ServicePrim
  AFBeginDialogueRsp (*! constructor *) :
    functional_units_Opt,
    tp_begin_dialogue_result,
    tp_begin_dialogue_diagnostic_Opt,
    correlator_Opt,
    mapping,
    PDUqueue (*user_data*)  -> ServicePrim
  AFBeginDialogueRsp (*! constructor *) :
    tp_begin_dialogue_result,
    tp_begin_channel_diagnostic_Opt,
    correlator_Opt,
    mapping -> ServicePrim
  AFBeginDialogueCnf (*! constructor *) :
    functional_units_Opt,
    tp_begin_dialogue_result,
    tp_begin_dialogue_diagnostic_Opt,
    correlator_Opt,
    mapping,
    PDUqueue (*user_data*)  -> ServicePrim
  AFBeginDialogueCnf (*! constructor *) :
    tp_begin_dialogue_result,
    tp_begin_channel_diagnostic_Opt,
    correlator_Opt,
    mapping -> ServicePrim
  AFBidReq (*! constructor *) :
    bool (*ccr_token_requested*),
    correlator_Opt (*last_partner_identifier*)      -> ServicePrim
```

```
AFBidInd (*! constructor *) :
  bool (*ccr_token_requested*),
  correlator_Opt (*last_partner_identifier*)      -> ServicePrim
AFBidRsp (*! constructor *) : tp_bid_result -> ServicePrim
AFBidCnf (*! constructor *) : tp_bid_result -> ServicePrim
AFEndDialogueReq (*! constructor *) : bool (*tp_end_dialogue_confirmation*)   -> ServicePrim
AFEndDialogueInd (*! constructor *) : bool (*tp_end_dialogue_confirmation*)   -> ServicePrim
AFEndDialogueRsp (*! constructor *) : -> ServicePrim
AFEndDialogueCnf (*! constructor *) : -> ServicePrim
AFUErrorReq (*! constructor *) : -> ServicePrim
AFUErrorInd (*! constructor *) : -> ServicePrim
AFUErrorRsp (*! constructor *) : -> ServicePrim
AFUErrorCnf (*! constructor *) : -> ServicePrim
AFAbortReq (*! constructor *) : tp_abort_type, mapping, PDUqueue (*user_data*)  -> ServicePrim
AFAbortReq (*! constructor *) : tp_abort_type, mapping, tp_P_abort_diagnostic -> ServicePrim
AFAbortInd (*! constructor *) : tp_abort_type, mapping, PDUqueue (*user_data*)  -> ServicePrim
AFAbortInd (*! constructor *) : tp_abort_type, mapping, tp_P_abort_diagnostic -> ServicePrim
AFGrantControlReq (*! constructor *) : -> ServicePrim
AFGrantControlInd (*! constructor *) : -> ServicePrim
AFRequestControlReq (*! constructor *) : -> ServicePrim
AFRequestControlInd (*! constructor *) : -> ServicePrim
AFHandshakeReq (*! constructor *) : confirmation_urgency_Opt -> ServicePrim
AFHandshakeInd (*! constructor *) : confirmation_urgency_Opt -> ServicePrim
AFHandshakeRsp (*! constructor *) : -> ServicePrim
AFHandshakeCnf (*! constructor *) : -> ServicePrim
AFHandshakeAndGrantControlReq (*! constructor *) : confirmation_urgency -> ServicePrim
AFHandshakeAndGrantControlInd (*! constructor *) : confirmation_urgency -> ServicePrim
AFHandshakeAndGrantControlRsp (*! constructor *) : -> ServicePrim
AFHandshakeAndGrantControlCnf (*! constructor *) : -> ServicePrim
AFDeferReq (*! constructor *) : tp_defer_type -> ServicePrim
AFDeferInd (*! constructor *) : tp_defer_type -> ServicePrim
AFPrepareReq (*! constructor *) : bool_Opt (*data_permitted_Opt*) -> ServicePrim
AFPrepareInd (*! constructor *) : bool_Opt (*data_permitted_Opt*) -> ServicePrim
AFHeuristicReportReq (*! constructor *) :
  mapping,
  heuristic_report,
  atomic_action_identifier_Opt,
  branch_identifier_Opt -> ServicePrim
AFHeuristicReportInd (*! constructor *) : mapping, heuristic_report -> ServicePrim
AFAbortAndHeuristicReportReq (*! constructor *) :
  mapping, heuristic_report, PDUqueue (*user_data*)  -> ServicePrim
AFAbortAndHeuristicReportInd (*! constructor *) :
  mapping, heuristic_report, PDUqueue (*user_data*)  -> ServicePrim
AFTokenPleaseReq (*! constructor *) : -> ServicePrim
AFTokenPleaseInd (*! constructor *) : -> ServicePrim
AFTokenGiveReq (*! constructor *) : reason, correlator_Opt -> ServicePrim
AFTokenGiveInd (*! constructor *) : reason, correlator_Opt -> ServicePrim
AFRecoverReq (*! constructor *) :
  mapping,
  recovery_context_handle,
  atomic_action_identifier,
  branch_identifier -> ServicePrim
AFRecoverInd (*! constructor *) : mapping, recovery_context_handle -> ServicePrim
```

```
AFBeginDialogue_req,
AFBeginDialogue_ind,
AFBeginDialogue_rsp,
AFBeginDialogue_cnf,
AFBid_req,
AFBid_ind,
AFBid_rsp,
AFBid_cnf,
AFEndDialogue_req,
AFEndDialogue_ind,
AFEndDialogue_rsp,
AFEndDialogue_cnf,
AFUError_req,
AFUError_ind,
AFUError_rsp,
AFUError_cnf,
AFAbort_req,
AFAbort_ind,
AFGrantControl_req,
AFGrantControl_ind,
AFRequestControl_req,
AFRequestControl_ind,
AFHandshake_req,
AFHandshake_ind,
AFHandshake_rsp,
AFHandshake_cnf,
AFHandshakeAndGrantControl_req,
AFHandshakeAndGrantControl_ind,
AFHandshakeAndGrantControl_rsp,
AFHandshakeAndGrantControl_cnf,
AFDefer_req,
AFDefer_ind,
AFPrepare_req,
AFPrepare_ind,
AFHeuristicReport_req,
AFHeuristicReport_ind,
AFAbortAndHeuristicReport_req,
AFAbortAndHeuristicReport_ind,
AFTokenPlease_req,
AFTokenPlease_ind,
AFTokenGive_req,
AFTokenGive_ind,
AFRecover_req,
AFRecover_ind,
first_AFsp,
last_AFsp : ->
    ServicePrimKey
IsAFBeginDialogueReq,
IsAFBeginDialogueInd,
IsAFBeginDialogueRsp,
IsAFBeginDialogueCnf,
IsAFBidReq,
IsAFBidInd,
IsAFBidRsp,
IsAFBidCnf,
IsAFEndDialogueReq,
IsAFEndDialogueInd,
IsAFEndDialogueRsp,
IsAFEndDialogueCnf,
IsAFUErrorReq,
IsAFUErrorInd,
IsAFUErrorRsp,
IsAFUErrorCnf,
IsAFAbortReq,
IsAFAbortInd,
IsAFGrantControlReq,
IsAFGrantControlInd,
IsAFRequestControlReq,
IsAFRequestControlInd,
IsAFHandshakeReq,
IsAFHandshakeInd,
IsAFHandshakeRsp,
IsAFHandshakeCnf,
IsAFHandshakeAndGrantControlReq,
IsAFHandshakeAndGrantControlInd,
IsAFHandshakeAndGrantControlRsp,
IsAFHandshakeAndGrantControlCnf,
IsAFDeferReq,
IsAFDeferInd,
IsAFPrepareReq,
IsAFPrepareInd,
IsAFHeuristicReportReq,
IsAFHeuristicReportInd,
IsAFAbortAndHeuristicReportReq,
IsAFAbortAndHeuristicReportInd,
IsAFTokenPleaseReq,
IsAFTokenPleaseInd,
IsAFTokenGiveReq,
IsAFTokenGiveInd,
IsAFRecoverReq,
IsAFRecoverInd,
is_dialogue,
is_channel,
IsAFsp :
  ServicePrim -> bool
get_type : ServicePrim -> tp_abort_type
get_bcdO : ServicePrim -> tp_begin_channel_diagnostic_Opt
get_br : ServicePrim -> tp_bid_result
get_ctr : ServicePrim -> bool (*ccr_token_requested*)
get_cu :
  ServicePrim -> channel_utilization
get_dccO : ServicePrim -> correlator_Opt
get_dt : ServicePrim -> tp_defer_type
get_lpiO : ServicePrim -> correlator_Opt (*last_partner_identifier*)
get_map :
  ServicePrim -> mapping
get_rch : ServicePrim -> recovery_context_handle
get_reason : ServicePrim -> reason
```

```
  get_aaid : ServicePrim -> atomic_action_identifier
  get_brid : ServicePrim -> branch_identifier
eqns
    forall
      bcdO : tp_begin_channel_diagnostic_Opt,
      bdc : tp_begin_dialogue_confirmation,
      bddO : tp_begin_dialogue_diagnostic_Opt,
      bdr : tp_begin_dialogue_result,
      br : tp_bid_result,
      btO : bool_Opt (*begin_transaction_Opt*),
      cnfu : confirmation_urgency,
      cnfuO : confirmation_urgency_Opt,
      ctr : bool (*ccr_token_requested*),
      cu : channel_utilization,
      dccO : correlator_Opt,
      dpO : bool_Opt (*data_permitted_Opt*),
      dt : tp_defer_type,
      edc : bool (*tp_end_dialogue_confirmation*),
      hr : heuristic_report,
      ittO :  (*initiating*) TPSU_title_Opt,
      lpiO : correlator_Opt (*last_partner_identifier*),
      map : mapping,
      pad : tp_P_abort_diagnostic,
      rch : recovery_context_handle,
      rttO :  (*recipient*) TPSU_title_Opt,
      sfu : functional_units,
      fuO : functional_units_Opt,
      tokr : reason,
      ud : PDUqueue (*user_data*),
      aaid : atomic_action_identifier,
      aaidO : atomic_action_identifier_Opt,
      brid : branch_identifier,
      bridO : branch_identifier_Opt,
      sp : ServicePrim
  ofsort ServicePrimKey
    first_AFsp = succ(last_TPsp);
    AFBeginDialogue_req = first_AFsp;
    AFBeginDialogue_ind = succ(AFBeginDialogue_req);
    AFBeginDialogue_rsp = succ(AFBeginDialogue_ind);
    AFBeginDialogue_cnf = succ(AFBeginDialogue_rsp);
    AFBid_req = succ(AFBeginDialogue_cnf);
    AFBid_ind = succ(AFBid_req);
    AFBid_rsp = succ(AFBid_ind);
    AFBid_cnf = succ(AFBid_rsp);
    AFEndDialogue_req = succ(AFBid_cnf);
    AFEndDialogue_ind = succ(AFEndDialogue_req);
    AFEndDialogue_rsp = succ(AFEndDialogue_ind);
    AFEndDialogue_cnf = succ(AFEndDialogue_rsp);
    AFUError_req = succ(AFEndDialogue_cnf);
    AFUError_ind = succ(AFUError_req);
    AFUError_rsp = succ(AFUError_ind);
    AFUError_cnf = succ(AFUError_rsp);
    AFAbort_req = succ(AFUError_cnf);
    AFAbort_ind = succ(AFAbort_req);
```

```
AFGrantControl_req = succ(AFAbort_ind);
AFGrantControl_ind = succ(AFGrantControl_req);
AFRequestControl_req = succ(AFGrantControl_ind);
AFRequestControl_ind = succ(AFRequestControl_req);
AFHandshake_req = succ(AFRequestControl_ind);
AFHandshake_ind = succ(AFHandshake_req);
AFHandshake_rsp = succ(AFHandshake_ind);
AFHandshake_cnf = succ(AFHandshake_rsp);
AFHandshakeAndGrantControl_req = succ(AFHandshake_cnf);
AFHandshakeAndGrantControl_ind = succ(AFHandshakeAndGrantControl_req);
AFHandshakeAndGrantControl_rsp = succ(AFHandshakeAndGrantControl_ind);
AFHandshakeAndGrantControl_cnf = succ(AFHandshakeAndGrantControl_rsp);
AFDefer_req = succ(AFHandshakeAndGrantControl_cnf);
AFDefer_ind = succ(AFDefer_req);
AFPrepare_req = succ(AFDefer_ind);
AFPrepare_ind = succ(AFPrepare_req);
AFHeuristicReport_req = succ(AFPrepare_ind);
AFHeuristicReport_ind = succ(AFHeuristicReport_req);
AFAbortAndHeuristicReport_req = succ(AFHeuristicReport_ind);
AFAbortAndHeuristicReport_ind = succ(AFAbortAndHeuristicReport_req);
AFTokenPlease_req = succ(AFAbortAndHeuristicReport_ind);
AFTokenPlease_ind = succ(AFTokenPlease_req);
AFTokenGive_req = succ(AFTokenPlease_ind);
AFTokenGive_ind = succ(AFTokenGive_req);
AFRecover_req = succ(AFTokenGive_ind);
AFRecover_ind = succ(AFRecover_req);
last_AFsp = AFRecover_ind;
key(AFBeginDialogueReq(itt0, rtt0, sfu, bt0, bdc, dcc0, lpi0, ud)) =
  AFBeginDialogue_req;
key(AFBeginDialogueReq(sfu, dcc0, cu, lpi0)) = AFBeginDialogue_req;
key(AFBeginDialogueInd(itt0, rtt0, sfu, bt0, bdc, dcc0, lpi0, ud)) =
  AFBeginDialogue_ind;
key(AFBeginDialogueInd(sfu, dcc0, cu, lpi0)) = AFBeginDialogue_ind;
key(AFBeginDialogueRsp(fu0, bdr, bdd0, dcc0, map, ud)) =
  AFBeginDialogue_rsp;
key(AFBeginDialogueRsp(bdr, bcd0, dcc0, map)) = AFBeginDialogue_rsp;
key(AFBeginDialogueCnf(fu0, bdr, bdd0, dcc0, map, ud)) =
  AFBeginDialogue_cnf;
key(AFBeginDialogueCnf(bdr, bcd0, dcc0, map)) = AFBeginDialogue_cnf;
key(AFBidReq(ctr, lpi0)) = AFBid_req;
key(AFBidInd(ctr, lpi0)) = AFBid_ind;
key(AFBidRsp(br)) = AFBid_rsp;
key(AFBidCnf(br)) = AFBid_cnf;
key(AFEndDialogueReq(edc)) = AFEndDialogue_req;
key(AFEndDialogueInd(edc)) = AFEndDialogue_ind;
key(AFEndDialogueRsp) = AFEndDialogue_rsp;
key(AFEndDialogueCnf) = AFEndDialogue_cnf;
key(AFUErrorReq) = AFUError_req;
key(AFUErrorInd) = AFUError_ind;
key(AFUErrorRsp) = AFUError_rsp;
key(AFUErrorCnf) = AFUError_cnf;
key(AFAbortReq(user, map, ud)) = AFAbort_req;
key(AFAbortReq(provider, map, pad)) = AFAbort_req;
key(AFAbortInd(user, map, ud)) = AFAbort_ind;
```

```
    key(AFAbortInd(provider, map, pad)) = AFAbort_ind;
    key(AFGrantControlReq) = AFGrantControl_req;
    key(AFGrantControlInd) = AFGrantControl_ind;
    key(AFRequestControlReq) = AFRequestControl_req;
    key(AFRequestControlInd) = AFRequestControl_ind;
    key(AFHandshakeReq(cnfu0)) = AFHandshake_req;
    key(AFHandshakeInd(cnfu0)) = AFHandshake_ind;
    key(AFHandshakeRsp) = AFHandshake_rsp;
    key(AFHandshakeCnf) = AFHandshake_cnf;
    key(AFHandshakeAndGrantControlReq(cnfu)) = AFHandshakeAndGrantControl_req;
    key(AFHandshakeAndGrantControlInd(cnfu)) = AFHandshakeAndGrantControl_ind;
    key(AFHandshakeAndGrantControlRsp) = AFHandshakeAndGrantControl_rsp;
    key(AFHandshakeAndGrantControlCnf) = AFHandshakeAndGrantControl_cnf;
    key(AFDeferReq(dt)) = AFDefer_req;
    key(AFDeferInd(dt)) = AFDefer_ind;
    key(AFPrepareReq(dp0)) = AFPrepare_req;
    key(AFPrepareInd(dp0)) = AFPrepare_ind;
    key(AFHeuristicReportReq(map, hr, aaid0, brid0)) = AFHeuristicReport_req;
    key(AFHeuristicReportInd(map, hr)) = AFHeuristicReport_ind;
    key(AFAbortAndHeuristicReportReq(map, hr, ud)) =
      AFAbortAndHeuristicReport_req;
    key(AFAbortAndHeuristicReportInd(map, hr, ud)) =
      AFAbortAndHeuristicReport_ind;
    key(AFTokenPleaseReq) = AFTokenPlease_req;
    key(AFTokenPleaseInd) = AFTokenPlease_ind;
    key(AFTokenGiveReq(tokr, dcc0)) = AFTokenGive_req;
    key(AFTokenGiveInd(tokr, dcc0)) = AFTokenGive_ind;
    key(AFRecoverReq(map, rch, aaid, brid)) = AFRecover_req;
    key(AFRecoverInd(map, rch)) = AFRecover_ind;
  ofsort bool
    IsAFBeginDialogueReq(sp) = sp eq AFBeginDialogue_req;
    IsAFBeginDialogueInd(sp) = sp eq AFBeginDialogue_ind;
    IsAFBeginDialogueRsp(sp) = sp eq AFBeginDialogue_rsp;
    IsAFBeginDialogueCnf(sp) = sp eq AFBeginDialogue_cnf;
    IsAFBidReq(sp) = sp eq AFBid_req;
    IsAFBidInd(sp) = sp eq AFBid_ind;
    IsAFBidRsp(sp) = sp eq AFBid_rsp;
    IsAFBidCnf(sp) = sp eq AFBid_cnf;
    IsAFEndDialogueReq(sp) = sp eq AFEndDialogue_req;
    IsAFEndDialogueInd(sp) = sp eq AFEndDialogue_ind;
    IsAFEndDialogueRsp(sp) = sp eq AFEndDialogue_rsp;
    IsAFEndDialogueCnf(sp) = sp eq AFEndDialogue_cnf;
    IsAFUErrorReq(sp) = sp eq AFUError_req;
    IsAFUErrorInd(sp) = sp eq AFUError_ind;
    IsAFUErrorRsp(sp) = sp eq AFUError_rsp;
    IsAFUErrorCnf(sp) = sp eq AFUError_cnf;
    IsAFAbortReq(sp) = sp eq AFAbort_req;
    IsAFAbortInd(sp) = sp eq AFAbort_ind;
    IsAFGrantControlReq(sp) = sp eq AFGrantControl_req;
    IsAFGrantControlInd(sp) = sp eq AFGrantControl_ind;
    IsAFRequestControlReq(sp) = sp eq AFRequestControl_req;
    IsAFRequestControlInd(sp) = sp eq AFRequestControl_ind;
    IsAFHandshakeReq(sp) = sp eq AFHandshake_req;
    IsAFHandshakeInd(sp) = sp eq AFHandshake_ind;
    IsAFHandshakeRsp(sp) = sp eq AFHandshake_rsp;
    IsAFHandshakeCnf(sp) = sp eq AFHandshake_cnf;
    IsAFHandshakeAndGrantControlReq(sp) = sp eq AFHandshakeAndGrantControl_req;
    IsAFHandshakeAndGrantControlInd(sp) = sp eq AFHandshakeAndGrantControl_ind;
    IsAFHandshakeAndGrantControlRsp(sp) = sp eq AFHandshakeAndGrantControl_rsp;
    IsAFHandshakeAndGrantControlCnf(sp) = sp eq AFHandshakeAndGrantControl_cnf;
    IsAFDeferReq(sp) = sp eq AFDefer_req;
    IsAFDeferInd(sp) = sp eq AFDefer_ind;
    IsAFPrepareReq(sp) = sp eq AFPrepare_req;
    IsAFPrepareInd(sp) = sp eq AFPrepare_ind;
    IsAFHeuristicReportReq(sp) = sp eq AFHeuristicReport_req;
    IsAFHeuristicReportInd(sp) = sp eq AFHeuristicReport_ind;
    IsAFAbortAndHeuristicReportReq(sp) = sp eq AFAbortAndHeuristicReport_req;
    IsAFAbortAndHeuristicReportInd(sp) = sp eq AFAbortAndHeuristicReport_ind;
    IsAFTokenPleaseReq(sp) = sp eq AFTokenPlease_req;
    IsAFTokenPleaseInd(sp) = sp eq AFTokenPlease_ind;
    IsAFTokenGiveReq(sp) = sp eq AFTokenGive_req;
    IsAFTokenGiveInd(sp) = sp eq AFTokenGive_ind;
    IsAFRecoverReq(sp) = sp eq AFRecover_req;
    IsAFRecoverInd(sp) = sp eq AFRecover_ind;
    IsAFsp(sp) = (first_AFsp le key(sp)) and (key(sp) le last_AFsp);
    is_dialogue(AFBeginDialogueReq(itt0, rtt0, sfu, bt0, bdc, dcc0, lpi0, ud)) =
      true;
    is_dialogue(AFBeginDialogueReq(sfu, dcc0, cu, lpi0)) = false;
    is_dialogue(AFBeginDialogueInd(itt0, rtt0, sfu, bt0, bdc, dcc0, lpi0, ud)) =
      true;
    is_dialogue(AFBeginDialogueInd(sfu, dcc0, cu, lpi0)) = false;
    is_dialogue(AFBeginDialogueRsp(fu0, bdr, bdd0, dcc0, map, ud)) = true;
    is_dialogue(AFBeginDialogueRsp(bdr, bcd0, dcc0, map)) = false;
    is_dialogue(AFBeginDialogueCnf(fu0, bdr, bdd0, dcc0, map, ud)) = true;
    is_dialogue(AFBeginDialogueCnf(bdr, bcd0, dcc0, map)) = false;
    is_channel(AFBeginDialogueReq(itt0, rtt0, sfu, bt0, bdc, dcc0, lpi0, ud)) =
      false;
    is_channel(AFBeginDialogueReq(sfu, dcc0, cu, lpi0)) = true;
    is_channel(AFBeginDialogueInd(itt0, rtt0, sfu, bt0, bdc, dcc0, lpi0, ud)) =
      false;
    is_channel(AFBeginDialogueInd(sfu, dcc0, cu, lpi0)) = true;
    is_channel(AFBeginDialogueRsp(fu0, bdr, bdd0, dcc0, map, ud)) = false;
    is_channel(AFBeginDialogueRsp(bdr, bcd0, dcc0, map)) = true;
    is_channel(AFBeginDialogueCnf(fu0, bdr, bdd0, dcc0, map, ud)) = false;
    is_channel(AFBeginDialogueCnf(bdr, bcd0, dcc0, map)) = true;
  ofsort tp_abort_type
    get_type(AFAbortReq(user, map, ud)) = user;
    get_type(AFAbortReq(provider, map, pad)) = provider;
    get_type(AFAbortInd(user, map, ud)) = user;
    get_type(AFAbortInd(provider, map, pad)) = provider;
  ofsort tp_begin_channel_diagnostic_Opt
    get_bcd0(AFBeginDialogueRsp(bdr, bcd0, dcc0, map)) = bcd0;
    get_bcd0(AFBeginDialogueCnf(bdr, bcd0, dcc0, map)) = bcd0;
  ofsort tp_begin_dialogue_confirmation
    get_bdc(AFBeginDialogueReq(itt0, rtt0, sfu, bt0, bdc, dcc0, lpi0, ud)) =
      bdc;
    get_bdc(AFBeginDialogueInd(itt0, rtt0, sfu, bt0, bdc, dcc0, lpi0, ud)) =
      bdc;
  ofsort tp_begin_dialogue_diagnostic_Opt
    get_bddO(AFBeginDialogueRsp(fuO, bdr, bddO, dccO, map, ud)) = bddO;
    get_bddO(AFBeginDialogueCnf(fuO, bdr, bddO, dccO, map, ud)) = bddO;
  ofsort tp_begin_dialogue_result
    get_bdr(AFBeginDialogueRsp(fuO, bdr, bddO, dccO, map, ud)) = bdr;
    get_bdr(AFBeginDialogueRsp(bdr, bcdO, dccO, map)) = bdr;
    get_bdr(AFBeginDialogueCnf(fuO, bdr, bddO, dccO, map, ud)) = bdr;
    get_bdr(AFBeginDialogueCnf(bdr, bcdO, dccO, map)) = bdr;
  ofsort tp_bid_result get_br(AFBidRsp(br)) = br; get_br(AFBidCnf(br)) = br;
  ofsort bool_Opt (*begin_transaction_Opt*)
    get_btO(
      AFBeginDialogueReq(ittO, rttO, sfu, btO, bdc, dccO, lpiO, ud)) =
      btO;
    get_btO(AFBeginDialogueInd(ittO, rttO, sfu, btO, bdc, dccO, lpiO, ud)) =
      btO;
  ofsort confirmation_urgency
    get_cnfu(AFHandshakeAndGrantControlReq(cnfu)) = cnfu;
    get_cnfu(AFHandshakeAndGrantControlInd(cnfu)) = cnfu;
  ofsort confirmation_urgency_Opt
    get_cnfuO(AFHandshakeReq(cnfuO)) = cnfuO;
    get_cnfuO(AFHandshakeInd(cnfuO)) = cnfuO;
  ofsort bool (*ccr_token_requested*)
    get_ctr(
      AFBidReq(ctr, lpiO)) =
      ctr;
    get_ctr(AFBidInd(ctr, lpiO)) = ctr;
  ofsort channel_utilization
    get_cu(AFBeginDialogueReq(sfu, dccO, cu, lpiO)) = cu;
    get_cu(AFBeginDialogueInd(sfu, dccO, cu, lpiO)) = cu;
  ofsort correlator_Opt
    get_dccO(AFBeginDialogueReq(ittO, rttO, sfu, btO, bdc, dccO, lpiO, ud)) =
      dccO;
    get_dccO(AFBeginDialogueReq(sfu, dccO, cu, lpiO)) = dccO;
    get_dccO(AFBeginDialogueInd(ittO, rttO, sfu, btO, bdc, dccO, lpiO, ud)) =
      dccO;
    get_dccO(AFBeginDialogueInd(sfu, dccO, cu, lpiO)) = dccO;
    get_dccO(AFBeginDialogueRsp(fuO, bdr, bddO, dccO, map, ud)) = dccO;
    get_dccO(AFBeginDialogueRsp(bdr, bcdO, dccO, map)) = dccO;
    get_dccO(AFBeginDialogueCnf(fuO, bdr, bddO, dccO, map, ud)) = dccO;
    get_dccO(AFBeginDialogueCnf(bdr, bcdO, dccO, map)) = dccO;
    get_dccO(AFTokenGiveReq(tokr, dccO)) = dccO;
    get_dccO(AFTokenGiveInd(tokr, dccO)) = dccO;
  ofsort reason
    get_reason(AFTokenGiveReq(tokr, dccO)) = tokr;
    get_reason(AFTokenGiveInd(tokr, dccO)) = tokr;
  ofsort
    bool_Opt
(*data_permitted_Opt*)

    get_dpO(AFPrepareReq(dpO)) = dpO; get_dpO(AFPrepareInd(dpO)) = dpO;
  ofsort tp_defer_type
    get_dt(AFDeferReq(dt)) = dt; get_dt(AFDeferInd(dt)) = dt;
  ofsort bool (*tp_end_dialogue_confirmation*)
    get_edc(
      AFEndDialogueReq(edc)) =
      edc;
    get_edc(AFEndDialogueInd(edc)) = edc;
  ofsort heuristic_report
    get_hr(AFHeuristicReportReq(map, hr, aaidO, bridO)) = hr;
    get_hr(AFHeuristicReportInd(map, hr)) = hr;
    get_hr(AFAbortAndHeuristicReportReq(map, hr, ud)) = hr;
    get_hr(AFAbortAndHeuristicReportInd(map, hr, ud)) = hr;
  ofsort  (*initiating*) TPSU_title_Opt
    get_ittO(AFBeginDialogueReq(ittO, rttO, sfu, btO, bdc, dccO, lpiO, ud)) =
      ittO;
    get_ittO(AFBeginDialogueInd(ittO, rttO, sfu, btO, bdc, dccO, lpiO, ud)) =
      ittO;
  ofsort correlator_Opt (*last_partner_identifier*)
    get_lpiO(
      AFBeginDialogueReq(ittO, rttO, sfu, btO, bdc, dccO, lpiO, ud)) =
      lpiO;
    get_lpiO(AFBeginDialogueReq(sfu, dccO, cu, lpiO)) = lpiO;
    get_lpiO(AFBeginDialogueInd(ittO, rttO, sfu, btO, bdc, dccO, lpiO, ud)) =
      lpiO;
    get_lpiO(AFBeginDialogueInd(sfu, dccO, cu, lpiO)) = lpiO;
    get_lpiO(AFBidReq(ctr, lpiO)) = lpiO;
    get_lpiO(AFBidInd(ctr, lpiO)) = lpiO;
  ofsort mapping
    get_map(AFBeginDialogueRsp(fuO, bdr, bddO, dccO, map, ud)) = map;
    get_map(AFBeginDialogueRsp(bdr, bcdO, dccO, map)) = map;
    get_map(AFBeginDialogueCnf(fuO, bdr, bddO, dccO, map, ud)) = map;
    get_map(AFBeginDialogueCnf(bdr, bcdO, dccO, map)) = map;
    get_map(AFAbortReq(user, map, ud)) = map;
    get_map(AFAbortReq(provider, map, pad)) = map;
    get_map(AFAbortInd(user, map, ud)) = map;
    get_map(AFAbortInd(provider, map, pad)) = map;
    get_map(AFHeuristicReportReq(map, hr, aaidO, bridO)) = map;
    get_map(AFHeuristicReportInd(map, hr)) = map;
    get_map(AFAbortAndHeuristicReportReq(map, hr, ud)) = map;
    get_map(AFAbortAndHeuristicReportInd(map, hr, ud)) = map;
    get_map(AFRecoverReq(map, rch, aaid, brid)) = map;
    get_map(AFRecoverInd(map, rch)) = map;
  ofsort tp_P_abort_diagnostic
    get_pad(AFAbortReq(provider, map, pad)) = pad;
    get_pad(AFAbortInd(provider, map, pad)) = pad;
  ofsort recovery_context_handle
    get_rch(AFRecoverReq(map, rch, aaid, brid)) = rch;
    get_rch(AFRecoverInd(map, rch)) = rch;
  ofsort  (*recipient*) TPSU_title_Opt
    get_rttO(AFBeginDialogueReq(ittO, rttO, sfu, btO, bdc, dccO, lpiO, ud)) =
      rttO;
    get_rttO(AFBeginDialogueInd(ittO, rttO, sfu, btO, bdc, dccO, lpiO, ud)) =
      rttO;
  ofsort functional_units
    get_sfu(AFBeginDialogueReq(ittO, rttO, sfu, btO, bdc, dccO, lpiO, ud)) =
      sfu;
    get_sfu(AFBeginDialogueReq(sfu, dccO, cu, lpiO)) = sfu;
    get_sfu(AFBeginDialogueInd(ittO, rttO, sfu, btO, bdc, dccO, lpiO, ud)) =
      sfu;
    get_sfu(AFBeginDialogueInd(sfu, dccO, cu, lpiO)) = sfu;
  ofsort functional_units_Opt
    get_fuO(AFBeginDialogueRsp(fuO, bdr, bddO, dccO, map, ud)) = fuO;
    get_fuO(AFBeginDialogueCnf(fuO, bdr, bddO, dccO, map, ud)) = fuO;
  ofsort PDUqueue (*user_data*)
    get_ud(
      AFBeginDialogueReq(ittO, rttO, sfu, btO, bdc, dccO, lpiO, ud)) =
      ud;
    get_ud(AFBeginDialogueInd(ittO, rttO, sfu, btO, bdc, dccO, lpiO, ud)) = ud;
    get_ud(AFBeginDialogueRsp(fuO, bdr, bddO, dccO, map, ud)) = ud;
    get_ud(AFBeginDialogueCnf(fuO, bdr, bddO, dccO, map, ud)) = ud;
    get_ud(AFAbortReq(user, map, ud)) = ud;
    get_ud(AFAbortInd(user, map, ud)) = ud;
    get_ud(AFAbortAndHeuristicReportReq(map, hr, ud)) = ud;
    get_ud(AFAbortAndHeuristicReportInd(map, hr, ud)) = ud;
  ofsort atomic_action_identifier
    get_aaid(AFRecoverReq(map, rch, aaid, brid)) = aaid;
    get_aaid(AFHeuristicReportReq(map, hr, Opt(aaid), bridO)) = aaid;
  ofsort branch_identifier
    get_brid(AFRecoverReq(map, rch, aaid, brid)) = brid;
    get_brid(AFHeuristicReportReq(map, hr, aaidO, Opt(brid))) = brid;
endtype
(* ------------------------------------------------------------------------
module AF_ServicePrim is UASE_ServicePrim

    type AF_ServicePrim is
        AFBeginDialogueReq (ITTO: TPSU_title_Opt,       /*initiating*/
                        RTTO: TPSU_title_Opt,   /*recipient*/
                        SFU: functional_units,
                        BTO: bool_Opt /*begin_transaction_Opt*/,
                        BDC: tp_begin_dialogue_confirmation,
                        DCCO: correlator_Opt,
                        LPIO: correlator_Opt,/*last_partner_identifier*/
                        UD: PDUqueue) /*user_data*/
    |   AFBeginDialogueReq (SFU: functional_units,
                        DCCO: correlator_Opt,
                        CU: channel_utilization,
                        LPIO: correlator_Opt) /*last_partner_identifier*/
    |   AFBeginDialogueInd (ITTO: TPSU_title_Opt,       /*initiating*/
                        RTTO: TPSU_title_Opt,   /*recipient*/
                        SFU: functional_units,
                        BTO: bool_Opt           /*begin_transaction_Opt*/,
                        BDC: tp_begin_dialogue_confirmation,
                        DCCO: correlator_Opt,
                        LPIO: correlator_Opt,   /*last_partner_identifier*/
                        UD: PDUqueue)           /*user_data*/
    |   AFBeginDialogueInd (SFU: functional_units,
                        DCCO: correlator_Opt,
                        CU: channel_utilization,
                        LPIO: correlator_Opt) /*last_partner_identifier*/
    |   AFBeginDialogueRsp (FUO: functional_units_Opt,
                        BDR: tp_begin_dialogue_result,
                        BDDO: tp_begin_dialogue_diagnostic_Opt,
                        DCCO: correlator_Opt,
                        MAP: mapping,
                        UD: PDUqueue)    /*user_data*/
        |   AFBeginDialogueRsp (BDR: tp_begin_dialogue_result,
                        BCDO: tp_begin_channel_diagnostic_Opt,
                        DCCO: correlator_Opt,
                        MAP: mapping)
        |   AFBeginDialogueCnf (FUO: functional_units_Opt,
                        BDR: tp_begin_dialogue_result,
                        BDDO: tp_begin_dialogue_diagnostic_Opt,
                        DCCO: correlator_Opt,
                        MAP: mapping,
                        UD: PDUqueue)    /*user_data*/
        |   AFBeginDialogueCnf (BDR: tp_begin_dialogue_result,
                        BCDO: tp_begin_channel_diagnostic_Opt,
                        DCCO: correlator_Opt,
                        MAP: mapping)
        |   AFBidReq (CTR: bool,     /*ccr_token_requested*/
                        LPIO: correlator_Opt) /*last_partner_identifier*/
        |   AFBidInd (CTR: bool,     /*ccr_token_requested*/
                        LPIO: correlator_Opt) /*last_partner_identifier*/
        |   AFBidRsp (BR: tp_bid_result)
        |   AFBidCnf (BR: tp_bid_result)
        |   AFEndDialogueReq (EDC: bool) /*tp_end_dialogue_confirmation*/
        |   AFEndDialogueInd (EDC: bool) /*tp_end_dialogue_confirmation*/
        |   AFEndDialogueRsp
        |   AFEndDialogueCnf
        |   AFUErrorReq
        |   AFUErrorInd
        |   AFUErrorRsp
        |   AFUErrorCnf
        |   AFAbortReq (AT: tp_abort_type, MAP: mapping, UD: PDUqueue)
        |   AFAbortReq (AT: tp_abort_type, MAP: mapping, PAD: tp_P_abort_diagnostic)
        |   AFAbortInd (AT: tp_abort_type, MAP: mapping, UD: PDUqueue)
        |   AFAbortInd (AT: tp_abort_type, MAP: mapping, PAD: tp_P_abort_diagnostic)
        |   AFGrantControlReq
        |   AFGrantControlInd
        |   AFRequestControlReq
        |   AFRequestControlInd
        |   AFHandshakeReq (CNFUO: confirmation_urgency_Opt)
        |   AFHandshakeInd (CNFUO: confirmation_urgency_Opt)
        |   AFHandshakeRsp
        |   AFHandshakeCnf
        |   AFHandshakeAndGrantControlReq (CNFU: confirmation_urgency)
        |   AFHandshakeAndGrantControlInd (CNFU: confirmation_urgency
        |   AFHandshakeAndGrantControlRsp
        |   AFHandshakeAndGrantControlCnf
        |   AFDeferReq (DT: tp_defer_type)
        |   AFDeferInd (DT: tp_defer_type)
        |   AFPrepareReq (DPO: bool_Opt) /*data_permitted_Opt*/
        |   AFPrepareInd (DPO: bool_Opt) /*data_permitted_Opt*/
        |   AFHeuristicReportReq (MAP: mapping,
                        HR: heuristic_report,
                        AAIDO: atomic_action_identifier_Opt,
                        BRIDO: branch_identifier_Opt)
        |   AFHeuristicReportInd (MAP: mapping, HR: heuristic_report)
        |   AFAbortAndHeuristicReportReq (MAP: mapping,
                        HR: heuristic_report,
                        UD: PDUqueue)    /*user_data*/
        |   AFAbortAndHeuristicReportInd (MAP: mapping,
                        HR: heuristic_report,
                        UD: PDUqueue)    /*user_data*/
        |   AFTokenPleaseReq
        |   AFTokenPleaseInd
        |   AFTokenGiveReq (TOKR: reason, DCCO: correlator_Opt)
        |   AFTokenGiveInd (TOKR: reason, DCCO: correlator_Opt)
        |   AFRecoverReq (MAP: mapping,
                        RCH: recovery_context_handle,
                        AAID: atomic_action_identifier,
                        BRID: branch_identifier)
        |   AFRecoverInd (MAP: mapping, RCH: recovery_context_handle)
    endtype
   /* no tralation is needed for AF*_{rsp|ind|cnf} */
   /* no trasnlation is needed for function key */
    function IsAFBeginDialogueReq (SP: AF_ServicePrim) : bool is
        SP match AFBeginDialogueReq (...) endfunc
    function IsAFBeginDialogueInd (SP: AF_ServicePrim) : bool is
        SP match AFBeginDialogueInd (...) endfunc
    function IsAFBeginDialogueRsp (SP: AF_ServicePrim) : bool is
        SP match AFBeginDialogueRsp (...) endfunc
    function IsAFBeginDialogueCnf (SP: AF_ServicePrim) : bool is
        SP match AFBeginDialogueCnf (...) endfunc
    function IsAFBidReq (SP: AF_ServicePrim) : bool is
        SP match AFBidReq (...) endfunc
    function IsAFBidInd (SP: AF_ServicePrim) : bool is
        SP match AFBidInd (...) endfunc
    function IsAFBidRsp (SP: AF_ServicePrim) : bool is
        SP match AFBidRsp (...) endfunc
    function IsAFBidCnf (SP: AF_ServicePrim) : bool is
        SP match AFBidCnf (...) endfunc
    function IsAFEndDialogueReq (SP: AF_ServicePrim) : bool is
        SP match AFEndDialogueReq (...) endfunc
    function IsAFEndDialogueInd (SP: AF_ServicePrim) : bool is
        SP match AFEndDialogueInd (...) endfunc
    function IsAFEndDialogueRsp (SP: AF_ServicePrim) : bool is
        SP match AFEndDialogueRsp endfunc
    function IsAFEndDialogueCnf (SP: AF_ServicePrim) : bool is
        SP match AFEndDialogueCnf endfunc
    function IsAFUErrorReq (SP: AF_ServicePrim) : bool is
        SP match AFUErrorReq endfunc
    function IsAFUErrorInd (SP: AF_ServicePrim) : bool is
        SP match AFUErrorInd endfunc
    function IsAFUErrorRsp (SP: AF_ServicePrim) : bool is
        SP match AFUErrorRsp endfunc
    function IsAFUErrorCnf (SP: AF_ServicePrim) : bool is
        SP match AFUErrorCnf endfunc
    function IsAFAbortReq (SP: AF_ServicePrim) : bool is
        SP match AFAbortReq (...) endfunc
```

```
function IsAFAbortInd (SP: AF_ServicePrim) : bool is
    SP match AFAbortInd (...) endfunc
function IsAFGrantControlReq (SP: AF_ServicePrim) : bool is
    SP match AFGrantControlReq endfunc
function IsAFGrantControlInd (SP: AF_ServicePrim) : bool is
    SP match AFGrantControlInd endfunc
function IsAFRequestControlReq (SP: AF_ServicePrim) : bool is
    SP match AFRequestControlReq endfunc
function IsAFRequestControlInd (SP: AF_ServicePrim) : bool is
    SP match AFRequestControlInd endfunc
function IsAFHandshakeReq (SP: AF_ServicePrim) : bool is
    SP match AFHandshakeReq (...) endfunc
function IsAFHandshakeInd (SP: AF_ServicePrim) : bool is
    SP match AFHandshakeInd (... endfunc
function IsAFHandshakeRsp (SP: AF_ServicePrim) : bool is
    SP match AFHandshakeRsp endfunc
function IsAFHandshakeCnf (SP: AF_ServicePrim) : bool is
    SP match AFHandshakeCnf endfunc
function IsAFHandshakeAndGrantControlReq (SP: AF_ServicePrim) : bool is
    SP match AFHandshakeAndGrantControlReq (...) endfunc
function IsAFHandshakeAndGrantControlInd (SP: AF_ServicePrim) : bool is
    SP match AFHandshakeAndGrantControlInd (...) endfunc
function IsAFHandshakeAndGrantControlRsp (SP: AF_ServicePrim) : bool is
    SP match AFHandshakeAndGrantControlRsp endfunc
function IsAFHandshakeAndGrantControlCnf (SP: AF_ServicePrim) : bool is
    SP match AFHandshakeAndGrantControlCnf endfunc
function IsAFDeferReq (SP: AF_ServicePrim) : bool is
    SP match AFDeferReq (...) endfunc
function IsAFDeferInd (SP: AF_ServicePrim) : bool is
    SP match AFDeferInd (...) endfunc
function IsAFPrepareReq (SP: AF_ServicePrim) : bool is
    SP match AFPrepareReq (...) endfunc
function IsAFPrepareInd (SP: AF_ServicePrim) : bool is
    SP match AFPrepareInd (...) endfunc
function IsAFHeuristicReportReq (SP: AF_ServicePrim) : bool is
    SP match AFHeuristicReportReq (...) endfunc
function IsAFHeuristicReportInd (SP: AF_ServicePrim) : bool is
    SP match AFHeuristicReportInd (...) endfunc
function IsAFAbortAndHeuristicReportReq (SP: AF_ServicePrim) : bool is
    SP match AFAbortAndHeuristicReportReq (...) endfunc
function IsAFAbortAndHeuristicReportInd (SP: AF_ServicePrim) : bool is
    SP match AFAbortAndHeuristicReportInd (...) endfunc
function IsAFTokenPleaseReq (SP: AF_ServicePrim) : bool is
    SP match AFTokenPleaseReq endfunc
function IsAFTokenPleaseInd (SP: AF_ServicePrim) : bool is
    SP match AFTokenPleaseInd endfunc
function IsAFTokenGiveReq (SP: AF_ServicePrim) : bool is
    SP match AFTokenGiveReq (...) endfunc
function IsAFTokenGiveInd (SP: AF_ServicePrim) : bool is
    SP match AFTokenGiveInd (...) endfunc
function IsAFRecoverReq (SP: AF_ServicePrim) : bool is
    SP match AFRecoverReq (...) endfunc
function IsAFRecoverInd (SP: AF_ServicePrim) : bool is
    SP match AFRecoverInd (...) endfunc
```

```
/*   IsAFsp(sp) = (first_AFsp le key(sp)) and (key(sp) le last_AFsp); */

    function is_dialogue (SP: AF_ServicePrim) : bool is
        case SP in
             AFBeginDialogueReq (ITTO:=I: TPSU_title_Opt,...)
        |    AFBeginDialogueInd (ITTO:=I: TPSU_title_Opt,...)
        |    AFBeginDialogueRsp (FUO:=F: functional_units_Opt,...)
        |    AFBeginDialogueCnf (FUO:=F: functional_units_Opt,...) -> true
        |    any AF_ServicePrim -> false
        endcase
    endfunc

    function is_channel (SP: AF_ServicePrim) : bool is
        case SP in
            AFBeginDialogueReq (SFU:=F functional_units,...)
        |   AFBeginDialogueInd (CU:=CU: channel_utilization,...)
        |   AFBeginDialogueRsp (bdr: tp_begin_dialogue_result,
                                bcdO: tp_begin_channel_diagnostic_Opt,
                                dccO: correlator_Opt,
                                map: mapping)
        |   AFBeginDialogueCnf (bdr: tp_begin_dialogue_result,
                                bcdO: tp_begin_channel_diagnostic_Opt,
                                dccO: correlator_Opt,
                                map: mapping) -> true
        |   any AF_ServicePrim -> false
        endcase
    endfunc

    function get_type (SP: AF_ServicePrim) : tp_abort_type is
        select AT in SP endfunc
    function get_bcdO (SP: AF_ServicePrim) : tp_begin_channel_diagnostic_Opt is
         select BCDO in SP endfunc
    function get_bdc (SP: AF_ServicePrim) : tp_begin_dialogue_confirmation is
        select BDC inSP endfunc
    function get_bddO (SP: AF_ServicePrim) : tp_begin_dialogue_diagnostic_Opt is
        select BDDO in SP endfunc
    function get_bdr (SP: AF_ServicePrim) : tp_begin_dialogue_result is
        select BDR in SP endfunc
    function get_br (SP: AF_ServicePrim) : tp_bid_result is
        select BR in SP endfunc
    function get_btO (SP: AF_ServicePrim) : bool_Opt is
        select BTO in SP endfunc
    function get_cnfu (SP: AF_ServicePrim) : confirmation_urgency is
        select CNFU in SP endfunc
    function get_cnfuO (SP: AF_ServicePrim) : confirmation_urgency_Opt is
        select CNFUO in SP endfunc
    function get_ctr (SP: AF_ServicePrim) : bool is
        select CTR in SP endfunc
    function get_cu (SP: AF_ServicePrim) : channel_utilization is
        select CU in SP endfunc
    function get_dccO (SP: AF_ServicePrim) : correlator_Opt is
        select DCCO in Sp endfunc
    function get_reason (SP: AF_ServicePrim) : reason is
        select TOKR in SP endfunc
    function get_dpO (SP: AF_ServicePrim) : bool_Opt is
        select DPO in SP endfunc
    function get_dt (SP: AF_ServicePrim) : tp_defer_type is
        select DT in SP endfunc
    function get_edc (SP: AF_ServicePrim) : bool is
        select EDC in SP endfunc
    function get_hr (SP: AF_ServicePrim) : heuristic_report is
        select HR in SP endfunc
    function get_ittO (SP: AF_ServicePrim) : TPSU_title_Opt is
        select ITTO in SP endfunc
    function get_lpiO (SP: AF_ServicePrim) : correlator_Opt is
        select LPIO in SP endfunc
    function get_map (SP: AF_ServicePrim) : mapping is
        select MAP in SP endfunc
    function get_pad (SP: AF_ServicePrim) : tp_P_abort_diagnostic is
        select PAD in SP endfunc
    function get_rch (SP: AF_ServicePrim) : recovery_context_handle is
        select RCH in SP endfunc
    function get_rttO (SP: AF_ServicePrim) : TPSU_title_Opt is
        select RTTO in SP endfunc
    function get_sfu (SP: AF_ServicePrim) : functional_units is
        select SFU in SP endfunc
    function get_fuO (SP: AF_ServicePrim) : functional_units_Opt is
        select FUO in SP endfunc
    function get_ud (SP: AF_ServicePrim) : PDUqueue is select UD in Sp endfunc
    function get_aaid (SP: AF_ServicePrim) : atomic_action_identifier is
        select AAID in SP endfunc
    function get_brid (SP: AF_ServicePrim) : branch_identifier is
        select BRID in SP endfunc
endmod
=========================================================================== *)

type AF_ServicePrim_extra_operations is AF_ServicePrim
opns
  set_dccO, set_lpiO : correlator_Opt, ServicePrim -> ServicePrim
  is_reserved_association : ServicePrim -> bool
  set_map : mapping, ServicePrim -> ServicePrim
  IsDeferGrantControl, IsDeferEndDialogue : ServicePrim -> bool
eqns
    forall
      dccO, dcc1 : correlator_Opt,
      lpiO, lpi1 : correlator_Opt (*last_partner_identifier*),
      map, map1 : mapping,
      bcdO : tp_begin_channel_diagnostic_Opt,
      bdc : tp_begin_dialogue_confirmation,
      bddO : tp_begin_dialogue_diagnostic_Opt,
      bdr : tp_begin_dialogue_result,
      btO : bool_Opt (*begin_transaction_Opt*),
      ctr : bool (*ccr_token_requested*),
      cu : channel_utilization,
      ittO :  (*initiating*) TPSU_title_Opt,
      rttO :  (*recipient*) TPSU_title_Opt,
      sfu : functional_units,
      fuO : functional_units_Opt,
      ud : PDUqueue (*user_data*),
      sp : ServicePrim
  ofsort bool
    is_reserved_association(sp) =
      (is_dialogue(sp) and is_association_reserved(get_bdd0(sp)))
      or
      (is_channel(sp) and is_association_reserved(get_bcd0(sp)));
  ofsort ServicePrim
    set_dcc0(
      dcc1, AFBeginDialogueReq(itt0, rtt0, sfu, bt0, bdc, dcc0, lpi0, ud)) =
      AFBeginDialogueReq(itt0, rtt0, sfu, bt0, bdc, dcc1, lpi0, ud);
    set_dcc0(dcc1, AFBeginDialogueReq(sfu, dcc0, cu, lpi0)) =
      AFBeginDialogueReq(sfu, dcc1, cu, lpi0);
    set_dcc0(dcc1, AFBeginDialogueRsp(fu0, bdr, bdd0, dcc0, map, ud)) =
      AFBeginDialogueRsp(fu0, bdr, bdd0, dcc1, map, ud);
    set_dcc0(dcc1, AFBeginDialogueRsp(bdr, bcd0, dcc0, map)) =
      AFBeginDialogueRsp(bdr, bcd0, dcc1, map);
    set_lpi0(
      lpi1, AFBeginDialogueReq(itt0, rtt0, sfu, bt0, bdc, dcc0, lpi0, ud)) =
      AFBeginDialogueReq(itt0, rtt0, sfu, bt0, bdc, dcc0, lpi1, ud);
    set_lpi0(lpi1, AFBeginDialogueReq(sfu, dcc0, cu, lpi0)) =
      AFBeginDialogueReq(sfu, dcc0, cu, lpi1);
    set_lpi0(lpi1, AFBidReq(ctr, lpi0)) = AFBidReq(ctr, lpi1);
    set_map(map1, AFBeginDialogueRsp(fu0, bdr, bdd0, dcc0, map, ud)) =
      AFBeginDialogueRsp(fu0, bdr, bdd0, dcc0, map1, ud);
    set_map(map1, AFBeginDialogueRsp(bdr, bcd0, dcc0, map)) =
      AFBeginDialogueRsp(bdr, bcd0, dcc0, map1);
  ofsort bool
    IsAFDeferReq(sp) or IsAFDeferInd(sp), get_dt(sp) = end_dialogue =>
      IsDeferEndDialogue(sp) = true;
    IsAFDeferReq(sp) or IsAFDeferInd(sp), get_dt(sp) = grant_control =>
      IsDeferEndDialogue(sp) = false;
    IsDeferGrantControl(sp) =
      (IsAFDeferReq(sp) or IsAFDeferInd(sp)) and not(IsDeferEndDialogue(sp));
endtype
(* ------------------------------------------------------------------------
module AF_ServicePrim_extra_operations is AF_ServicePrim

    function is_reserved_association (SP: AF_ServicePrim) : bool is
      (is_dialogue(sp) and is_association_reserved(get_bdd0(sp))) or
      (is_channel(sp) and is_association_reserved(get_bcd0(sp)))
    endfunc

    function set_dcc0 (dcc1: correlator_Opt, SP: AF_ServicePrim) : AF_ServicePrim is
        SP.{DCC0:=dcc1} endfunc

    function set_lpi0 (lpi1: correlator_Opt, SP: AF_ServicePrim) : AF_ServicePrim is
        SP.{LPI0:=lpi1} endfunc

    function set_map (map1: mapping, SP: AF_ServicePrim) : AF_ServicePrim is
        SP.{MAP:=map1} endfunc

    function IsDeferEndDialogue (SP: AF_ServicePrim) : bool is
        (IsAFDeferReq(SP) or IsAFDeferInd(SP)) and
        (get_dt(SP) eq end_dialogue)
    endfunc

    function IsDeferGrantControl (SP: AF_ServicePrim) : bool is
        (IsAFDeferReq(SP) or IsAFDeferInd(SP)) and
        (get_dt (SP) eq grant_control)
    endfunc
endmod
========================================================================== *)

type CAF_ServicePrim is AF_ServicePrim_extra_operations
opns
  CAFPleaseReq (*! constructor *) :
    AE_title, atomic_action_identifier, branch_identifier -> ServicePrim
  CAFGiveInd (*! constructor *) : channel_utilization -> ServicePrim
  CAFDetachReq (*! constructor *) : detach_type -> ServicePrim
  CAFRecoverInd (*! constructor *) :
    channel_utilization,
    recovery_state,
    atomic_action_identifier,
    branch_identifier ->
      ServicePrim
  CAFFailInd (*! constructor *) : -> ServicePrim

  CAFPlease_req,
  CAFGive_ind,
  CAFDetach_req,
  CAFRecover_ind,
  CAFFail_ind,
  first_CAFsp,
  last_CAFsp : -> ServicePrimKey
  IsCAFPleaseReq,
  IsCAFGiveInd,
  IsCAFDetachReq,
  IsCAFRecoverInd,
  IsCAFFailInd,
  IsCAFsp : ServicePrim -> bool
  get_aet : ServicePrim -> AE_title
  get_rsri : ServicePrim -> recovery_state
  get_dt : ServicePrim -> detach_type
eqns
    forall
      cu : channel_utilization,
      aet : AE_title,
      aaid : atomic_action_identifier,
      brid : branch_identifier,
      rsri : recovery_state,
      dt : detach_type,
      sp : ServicePrim
  ofsort ServicePrimKey
    first_CAFsp = succ(last_AFsp);
    CAFPlease_req = first_CAFsp;
    CAFGive_ind = succ(CAFPlease_req);
    CAFDetach_req = succ(CAFGive_ind);
    CAFRecover_ind = succ(CAFDetach_req);
    CAFFail_ind = succ(CAFRecover_ind);
    last_CAFsp = CAFFail_ind;
    key(CAFPleaseReq(aet, aaid, brid)) = CAFPlease_req;
    key(CAFGiveInd(cu)) = CAFGive_ind;
    key(CAFDetachReq(dt)) = CAFDetach_req;
    key(CAFRecoverInd(cu, rsri, aaid, brid)) = CAFRecover_ind;
    key(CAFFailInd) = CAFFail_ind;
  ofsort bool
    IsCAFPleaseReq(sp) = sp eq CAFPlease_req;
    IsCAFGiveInd(sp) = sp eq CAFGive_ind;
    IsCAFDetachReq(sp) = sp eq CAFDetach_req;
    IsCAFRecoverInd(sp) = sp eq CAFRecover_ind;
    IsCAFFailInd(sp) = sp eq CAFFail_ind;
    IsCAFsp(sp) = (first_CAFsp le key(sp)) and (key(sp) le last_CAFsp);
  ofsort detach_type get_dt(CAFDetachReq(dt)) = dt;
  ofsort channel_utilization
    get_cu(CAFGiveInd(cu)) = cu;
    get_cu(CAFRecoverInd(cu, rsri, aaid, brid)) = cu;
  ofsort AE_title get_aet(CAFPleaseReq(aet, aaid, brid)) = aet;
  ofsort atomic_action_identifier
    get_aaid(CAFPleaseReq(aet, aaid, brid)) = aaid;
    get_aaid(CAFRecoverInd(cu, rsri, aaid, brid)) = aaid;
  ofsort branch_identifier
    get_brid(CAFPleaseReq(aet, aaid, brid)) = brid;
    get_brid(CAFRecoverInd(cu, rsri, aaid, brid)) = brid;
  ofsort recovery_state get_rsri(CAFRecoverInd(cu, rsri, aaid, brid)) = rsri;
endtype
(* ----------------------------------------------------------------------------
module CAF_ServicePrim is AF_ServicePrim_extra_operations

    type CAF_ServicePrim is
        CAFPleaseReq (AET: AE_title,
                AAID: atomic_action_identifier,
                BRID: branch_identifier)
    |   CAFGiveInd (CU: channel_utilization)
    |   CAFDetachReq (DT: detach_type)
    |   CAFRecoverInd (CU: channel_utilization,
                RSRI: recovery_state,
                AAID: atomic_action_identifier,
                BRID: branch_identifier)
    |   CAFFailInd
    endtype

/* no translation is needed for CAF*_{req|ind} */
/* no translation is needed for key */
    function IsCAFPleaseReq (SP: CAF_ServicePrim) : bool is
        SP match CAFPleaseReq (...) endfunc
    function IsCAFGiveInd (SP: CAF_ServicePrim) : bool is
        SP match CAFGive_ind (...) endfunc
    function IsCAFDetachReq (SP: CAF_ServicePrim) : bool is
        SP match CAFDetach_req (...) endfunc
    function IsCAFRecoverInd (SP: CAF_ServicePrim) : bool is
        SP match CAFRecover_ind (...) endfunc
    function IsCAFFailInd (SP: CAF_ServicePrim) : bool is
        SP match CAFFail_ind (...) endfunc
/* IsCAFsp(sp) = (first_CAFsp le key(sp)) and (key(sp) le last_CAFsp); */

    function get_dt (SP: CAF_ServicePrim) : detach_type is
        select DT in SP endfunc
    function get_cu (SP: CAF_ServicePrim) : channel_utilization is
        select CU in SP endfunc
    function get_aet (SP: CAF_ServicePrim) : AE_title is
        select AET in Sp endfunc
    function get_aaid (SP: CAF_ServicePrim) : atomic_action_identifier is
        select AAID in SP endfunc
    function get_brid (SP: CAF_ServicePrim) : branch_identifier is
        select BRID in SP endfunc
    function get_rsri (SP: CAF_ServicePrim) : recovery_state is
        select RSRI in SP endfunc
endmod
============================================================================ *)

type SAF_ServicePrim is CAF_ServicePrim
opns
  SAFDetachAssociationReq (*! constructor *) : status -> ServicePrim
  SAFAssociationLostInd (*! constructor *) : -> ServicePrim
  SAFDetachAssociation_req, SAFAssociationLost_ind, first_SAFsp, last_SAFsp : ->
      ServicePrimKey
  IsSAFDetachAssociationReq, IsSAFAssociationLostInd, IsSAFsp :
    ServicePrim -> bool
  get_status : ServicePrim -> status
eqns
    forall status : status, sp : ServicePrim
  ofsort ServicePrimKey
    first_SAFsp = succ(last_CAFsp);
    SAFDetachAssociation_req = first_SAFsp;
    SAFAssociationLost_ind = succ(SAFDetachAssociation_req);
    last_SAFsp = SAFAssociationLost_ind;
    key(SAFDetachAssociationReq(status)) = SAFDetachAssociation_req;
    key(SAFAssociationLostInd) = SAFAssociationLost_ind;
  ofsort bool
    IsSAFDetachAssociationReq(sp) = sp eq SAFDetachAssociation_req;
    IsSAFAssociationLostInd(sp) = sp eq SAFAssociationLost_ind;
    IsSAFsp(sp) = (first_SAFsp le key(sp)) and (key(sp) le last_SAFsp);
  ofsort status get_status(SAFDetachAssociationReq(status)) = status;
endtype
(* -------------------------------------------------------------------------
module SAF_ServicePrim is CAF_ServicePrim

    type SAF_ServicePrim is
        SAFDetachAssociationReq (S: status)
    |   SAFAssociationLostInd
    endtype

/* no trasnlation is needed for SAF*_{req|ind} */
/* no translation is needed for key */
```

```
    function IsSAFDetachAssociationReq (SP: SAF_ServicePrim) : bool is
        SP match SAFDetachAssociationReq (...) endfunc
    function IsSAFAssociationLostInd (SP: SAF_ServicePrim) : bool is
        SP match SAFAssociationLostInd endfunc
/*    IsSAFsp(sp) = (first_SAFsp le key(sp)) and (key(sp) le last_SAFsp); */
    function get_status (SP: SAF_ServicePrim) : status is select S in SP endfunc
endmod
=========================================================================== *)

type CCR_ServicePrim is SAF_ServicePrim
opns
  CBeginReq (*! constructor *) :
              (*atomic_action_identifier_masters_name*) AE_title,
              (*atomic_action_identifier*) suffix,
              (*branch_identifier_superiors_name*) AE_title,
              (*branch_identifier49z*) suffix -> ServicePrim
  CCommit_CBeginReq (*! constructor *) :
              (*atomic_action_identifier_masters_name*) AE_title,
              (*atomic_action_identifier*) suffix,
              (*branch_identifier_superiors_name*) AE_title,
              (*branch_identifier*) suffix -> ServicePrim
  CBeginInd (*! constructor *) :
              (*atomic_action_identifier_masters_name*) AE_title,
              (*atomic_action_identifier*) suffix,
              (*branch_identifier_superiors_name*) AE_title,
              (*branch_identifier*) suffix -> ServicePrim
  CCommit_CBeginInd (*! constructor *) :
              (*atomic_action_identifier_masters_name*) AE_title,
              (*atomic_action_identifier*) suffix,
              (*branch_identifier_superiors_name*) AE_title,
              (*branch_identifier*) suffix -> ServicePrim
  CBeginRsp (*! constructor *) : -> ServicePrim
  CBeginCnf (*! constructor *) : -> ServicePrim
  CPrepareReq (*! constructor *) : PDUqueue (*user_data*)  -> ServicePrim
  CPrepareInd (*! constructor *) : PDUqueue (*user_data*)  -> ServicePrim
  CReadyReq (*! constructor *) : -> ServicePrim
  CReadyInd (*! constructor *) : -> ServicePrim
  CCommitReq (*! constructor *) : PDUqueue (*user_data*)  -> ServicePrim
  CCommitInd (*! constructor *) : PDUqueue (*user_data*)  -> ServicePrim
  CCommitRsp (*! constructor *) : PDUqueue (*user_data*)  -> ServicePrim
  CCommitCnf (*! constructor *) : PDUqueue (*user_data*)  -> ServicePrim
  CRollbackReq (*! constructor *) : PDUqueue (*user_data*)  -> ServicePrim
  CRollbackInd (*! constructor *) : PDUqueue (*user_data*)  -> ServicePrim
  CRollbackRsp (*! constructor *) : PDUqueue (*user_data*)  -> ServicePrim
  CRollbackCnf (*! constructor *) : PDUqueue (*user_data*)  -> ServicePrim
  CRecoverReq (*! constructor *) :
    recovery_state,
    atomic_action_identifier,
    branch_identifier,
    PDUqueue (*user_data*)  -> ServicePrim
  CRecoverInd (*! constructor *) :
    recovery_state,
    atomic_action_identifier,
    branch_identifier,
    PDUqueue (*user_data*)  -> ServicePrim
  CRecoverRsp (*! constructor *) :
    recovery_response (*recovery_state*),
    atomic_action_identifier,
    branch_identifier,
    PDUqueue (*user_data*)  -> ServicePrim
  CRecoverCnf (*! constructor *) :
    recovery_response (*recovery_state*),
    atomic_action_identifier,
    branch_identifier,
    PDUqueue (*user_data*)  -> ServicePrim

  CBegin_req,
  CCommit_CBegin_req,
  CBegin_ind,
  CCommit_CBegin_ind,
  CBegin_rsp,
  CBegin_cnf,
  CPrepare_req,
  CPrepare_ind,
  CReady_req,
  CReady_ind,
  CCommit_req,
  CCommit_ind,
  CCommit_rsp,
  CCommit_cnf,
  CRollback_req,
  CRollback_ind,
  CRollback_rsp,
  CRollback_cnf,
  CRecover_req,
  CRecover_ind,
  CRecover_rsp,
  CRecover_cnf,
  first_CCRsp,
  last_CCRsp : ->
      ServicePrimKey
  IsCBeginReq,
  IsCCommit_CBeginReq,
  IsCBeginInd,
  IsCCommit_CBeginInd,
  IsCBeginRsp,
  IsCBeginCnf,
  IsCPrepareReq,
  IsCPrepareInd,
  IsCReadyReq,
  IsCReadyInd,
  IsCCommitReq,
  IsCCommitInd,
  IsCCommitRsp,
  IsCCommitCnf,
  IsCRollbackReq,
  IsCRollbackInd,
  IsCRollbackRsp,
  IsCRollbackCnf,
  IsCRecoverReq,
  IsCRecoverInd,
  IsCRecoverRsp,
  IsCRecoverCnf,
  IsCCRsp :
    ServicePrim -> bool
  get_aaimn :
    ServicePrim -> (*atomic_action_identifier_masters_name*) AE_title
  get_aais : ServicePrim -> (*atomic_action_identifier*) suffix
  get_brisn : ServicePrim -> (*branch_identifier_superiors_name*) AE_title
  get_bris : ServicePrim -> (*branch_identifier*) suffix
  get_rsrc : ServicePrim -> recovery_response (*recovery_state*)
eqns
    forall
      aaid : atomic_action_identifier,
      aaimn : (*atomic_action_identifier_masters_name*) AE_title,
      aais :  (*atomic_action_identifier*) suffix,
      brid : branch_identifier,
      brisn : (*branch_identifier_superiors_name*) AE_title,
      bris :  (*branch_identifier*) suffix,
      ud : PDUqueue (*user_data*),
      rsri : recovery_state,
      rsrc : recovery_response (*recovery_state*),
      sp : ServicePrim
  ofsort ServicePrimKey
    first_CCRsp = succ(last_SAFsp);
    CBegin_req = first_CCRsp;
    CCommit_CBegin_req = succ(CBegin_req);
    CBegin_ind = succ(CCommit_CBegin_req);
    CCommit_CBegin_ind = succ(CBegin_ind);
    CBegin_rsp = succ(CCommit_CBegin_ind);
    CBegin_cnf = succ(CBegin_rsp);
    CPrepare_req = succ(CBegin_cnf);
    CPrepare_ind = succ(CPrepare_req);
    CReady_req = succ(CPrepare_ind);
    CReady_ind = succ(CReady_req);
    CCommit_req = succ(CReady_ind);
    CCommit_ind = succ(CCommit_req);
    CCommit_rsp = succ(CCommit_ind);
    CCommit_cnf = succ(CCommit_rsp);
    CRollback_req = succ(CCommit_cnf);
    CRollback_ind = succ(CRollback_req);
    CRollback_rsp = succ(CRollback_ind);
    CRollback_cnf = succ(CRollback_rsp);
    CRecover_req = succ(CRollback_cnf);
    CRecover_ind = succ(CRecover_req);
    CRecover_rsp = succ(CRecover_ind);
    CRecover_cnf = succ(CRecover_rsp);
    last_CCRsp = CRecover_cnf;
    key(CBeginReq(aaimn, aais, brisn, bris)) = CBegin_req;
    key(CCommit_CBeginReq(aaimn, aais, brisn, bris)) = CCommit_CBegin_req;
    key(CBeginInd(aaimn, aais, brisn, bris)) = CBegin_ind;
    key(CCommit_CBeginInd(aaimn, aais, brisn, bris)) = CCommit_CBegin_ind;
    key(CBeginRsp) = CBegin_rsp;
    key(CBeginCnf) = CBegin_cnf;
    key(CPrepareReq(ud)) = CPrepare_req;
    key(CPrepareInd(ud)) = CPrepare_ind;
    key(CReadyReq) = CReady_req;
    key(CReadyInd) = CReady_ind;
    key(CCommitReq(ud)) = CCommit_req;
    key(CCommitInd(ud)) = CCommit_ind;
    key(CCommitRsp(ud)) = CCommit_rsp;
    key(CCommitCnf(ud)) = CCommit_cnf;
    key(CRollbackReq(ud)) = CRollback_req;
    key(CRollbackInd(ud)) = CRollback_ind;
    key(CRollbackRsp(ud)) = CRollback_rsp;
    key(CRollbackCnf(ud)) = CRollback_cnf;
    key(CRecoverReq(rsri, aaid, brid, ud)) = CRecover_req;
    key(CRecoverInd(rsri, aaid, brid, ud)) = CRecover_ind;
    key(CRecoverRsp(rsrc, aaid, brid, ud)) = CRecover_rsp;
    key(CRecoverCnf(rsrc, aaid, brid, ud)) = CRecover_cnf;
  ofsort bool
    IsCBeginReq(sp) = sp eq CBegin_req;
    IsCCommit_CBeginReq(sp) = sp eq CCommit_CBegin_req;
    IsCBeginInd(sp) = sp eq CBegin_ind;
    IsCCommit_CBeginInd(sp) = sp eq CCommit_CBegin_ind;
    IsCBeginRsp(sp) = sp eq CBegin_rsp;
    IsCBeginCnf(sp) = sp eq CBegin_cnf;
    IsCPrepareReq(sp) = sp eq CPrepare_req;
    IsCPrepareInd(sp) = sp eq CPrepare_ind;
    IsCReadyReq(sp) = sp eq CReady_req;
    IsCReadyInd(sp) = sp eq CReady_ind;
    IsCCommitReq(sp) = sp eq CCommit_req;
    IsCCommitInd(sp) = sp eq CCommit_ind;
    IsCCommitRsp(sp) = sp eq CCommit_rsp;
    IsCCommitCnf(sp) = sp eq CCommit_cnf;
    IsCRollbackReq(sp) = sp eq CRollback_req;
    IsCRollbackInd(sp) = sp eq CRollback_ind;
    IsCRollbackRsp(sp) = sp eq CRollback_rsp;
    IsCRollbackCnf(sp) = sp eq CRollback_cnf;
    IsCRecoverReq(sp) = sp eq CRecover_req;
    IsCRecoverInd(sp) = sp eq CRecover_ind;
    IsCRecoverRsp(sp) = sp eq CRecover_rsp;
    IsCRecoverCnf(sp) = sp eq CRecover_cnf;
    IsCCRsp(sp) = (first_CCRsp le key(sp)) and (key(sp) le last_CCRsp);
  ofsort atomic_action_identifier
    get_aaid(CRecoverReq(rsri, aaid, brid, ud)) = aaid;
    get_aaid(CRecoverInd(rsri, aaid, brid, ud)) = aaid;
    get_aaid(CRecoverRsp(rsrc, aaid, brid, ud)) = aaid;
    get_aaid(CRecoverCnf(rsrc, aaid, brid, ud)) = aaid;
  ofsort  (*atomic_action_identifier_masters_name*) AE_title
    get_aaimn(CBeginReq(aaimn, aais, brisn, bris)) = aaimn;
    get_aaimn(CCommit_CBeginReq(aaimn, aais, brisn, bris)) = aaimn;
    get_aaimn(CBeginInd(aaimn, aais, brisn, bris)) = aaimn;
    get_aaimn(CCommit_CBeginInd(aaimn, aais, brisn, bris)) = aaimn;
  ofsort  (*atomic_action_identifier*) suffix
    get_aais(CBeginReq(aaimn, aais, brisn, bris)) = aais;
    get_aais(CCommit_CBeginReq(aaimn, aais, brisn, bris)) = aais;
    get_aais(CBeginInd(aaimn, aais, brisn, bris)) = aais;
    get_aais(CCommit_CBeginInd(aaimn, aais, brisn, bris)) = aais;
  ofsort branch_identifier
    get_brid(CRecoverReq(rsri, aaid, brid, ud)) = brid;
    get_brid(CRecoverInd(rsri, aaid, brid, ud)) = brid;
    get_brid(CRecoverRsp(rsrc, aaid, brid, ud)) = brid;
    get_brid(CRecoverCnf(rsrc, aaid, brid, ud)) = brid;
  ofsort  (*branch_identifier_superiors_name*) AE_title
    get_brisn(CBeginReq(aaimn, aais, brisn, bris)) = brisn;
    get_brisn(CCommit_CBeginReq(aaimn, aais, brisn, bris)) = brisn;
    get_brisn(CBeginInd(aaimn, aais, brisn, bris)) = brisn;
    get_brisn(CCommit_CBeginInd(aaimn, aais, brisn, bris)) = brisn;
  ofsort  (*branch_identifier*) suffix
    get_bris(CBeginReq(aaimn, aais, brisn, bris)) = bris;
    get_bris(CCommit_CBeginReq(aaimn, aais, brisn, bris)) = bris;
    get_bris(CBeginInd(aaimn, aais, brisn, bris)) = bris;
    get_bris(CCommit_CBeginInd(aaimn, aais, brisn, bris)) = bris;
  ofsort PDUqueue (*user_data*)
    get_ud(
      CPrepareReq(ud)) =
      ud;
    get_ud(CPrepareInd(ud)) = ud;
    get_ud(CCommitReq(ud)) = ud;
    get_ud(CCommitInd(ud)) = ud;
    get_ud(CCommitRsp(ud)) = ud;
    get_ud(CCommitCnf(ud)) = ud;
    get_ud(CRollbackReq(ud)) = ud;
    get_ud(CRollbackInd(ud)) = ud;
    get_ud(CRollbackRsp(ud)) = ud;
    get_ud(CRollbackCnf(ud)) = ud;
    get_ud(CRecoverReq(rsri, aaid, brid, ud)) = ud;
    get_ud(CRecoverInd(rsri, aaid, brid, ud)) = ud;
    get_ud(CRecoverRsp(rsrc, aaid, brid, ud)) = ud;
    get_ud(CRecoverCnf(rsrc, aaid, brid, ud)) = ud;
  ofsort recovery_state
    get_rsri(CRecoverReq(rsri, aaid, brid, ud)) = rsri;
    get_rsri(CRecoverInd(rsri, aaid, brid, ud)) = rsri;
  ofsort recovery_response (*recovery_state*)
    get_rsrc(
      CRecoverRsp(rsrc, aaid, brid, ud)) =
      rsrc;
    get_rsrc(CRecoverCnf(rsrc, aaid, brid, ud)) = rsrc;
endtype
(* ------------------------------------------------------------------------
module CCR_ServicePrim is SAF_ServicePrim

    type CCR_ServicePrim is
        CBeginReq (AAIMN: AE_title, /*atomic_action_identifier_masters_name*/
                AAIS: suffix,           /*atomic_action_identifier*/
                BRISN:  AE_title,       /*branch_identifier_superiors_name*/
                bris: suffix)           /*branch_identifier*/
    |   CCommit_CBeginReq (AAIMN: AE_title, /*atomic_action_identifier_masters_name*/
                AAIS: suffix,           /*atomic_action_identifier*/
                BRISN:  AE_title,       /*branch_identifier_superiors_name*/
                bris: suffix)           /*branch_identifier*/
    |   CBeginInd (AAIMN: AE_title, /*atomic_action_identifier_masters_name*/
                AAIS: suffix,           /*atomic_action_identifier*/
                BRISN:  AE_title,       /*branch_identifier_superiors_name*/
                bris: suffix)           /*branch_identifier*/
    |   CCommit_CBeginInd (AAIMN: AE_title, /*atomic_action_identifier_masters_name*/
                AAIS: suffix,           /*atomic_action_identifier*/
                BRISN:  AE_title,       /*branch_identifier_superiors_name*/
                bris: suffix)           /*branch_identifier*/
    |   CBeginRsp
    |   CBeginCnf
    |   CPrepareReq (UD: PDUqueue)      /*user_data*/
    |   CPrepareInd (UD: PDUqueue)      /*user_data*/
    |   CReadyReq
    |   CReadyInd
    |   CCommitReq (UD: PDUqueue)       /*user_data*/
    |   CCommitInd (UD: PDUqueue)       /*user_data*/
    |   CCommitRsp (UD: PDUqueue)       /*user_data*/
    |   CCommitCnf (UD: PDUqueue)       /*user_data*/
    |   CRollbackReq (UD: PDUqueue)     /*user_data*/
    |   CRollbackInd (UD: PDUqueue)     /*user_data*/
    |   CRollbackRsp (UD: PDUqueue)     /*user_data*/
    |   CRollbackCnf (UD: PDUqueue)     /*user_data*/
    |   CRecoverReq (RSRI: recovery_state,
                AAID: atomic_action_identifier,
                BRID: branch_identifier,
                UD: PDUqueue)   /*user_data*/
    |   CRecoverInd (RSRI: recovery_state,
                AAID: atomic_action_identifier,
                BRID: branch_identifier,
                UD: PDUqueue)   /*user_data*/
    |   CRecoverRsp (RSRC: recovery_response,
                AAID: atomic_action_identifier,
                BRID: branch_identifier,
                UD: PDUqueue)   /*user_data*/
    |   CRecoverCnf (RSRC: recovery_response,
                AAID: atomic_action_identifier,
                BRID: branch_identifier,
                UD: PDUqueue)   /*user_data*/
    endtype

/* no trasnlation is needed for C*_{rsp|ind|req|cnf} */
/* no translation is needed for key */

    function IsCBeginReq (SP: CCR_ServicePrim) : bool is SP match CBeginReq endfunc
    function IsCCommit_CBeginReq (SP: CCR_ServicePrim) : bool is SP match CCommit_CBeginReq endfunc
    function IsCBeginInd (SP: CCR_ServicePrim) : bool is SP match CBeginInd endfunc
    function IsCCommit_CBeginInd (SP: CCR_ServicePrim) : bool is SP match CCommit_CBeginInd endfunc
    function IsCBeginRsp (SP: CCR_ServicePrim) : bool is SP match CBeginRsp endfunc
    function IsCBeginCnf (SP: CCR_ServicePrim) : bool is SP match CBeginCnf endfunc
    function IsCPrepareReq (SP: CCR_ServicePrim) : bool is SP match CPrepareReq endfunc
    function IsCPrepareInd (SP: CCR_ServicePrim) : bool is SP match CPrepareInd endfunc
    function IsCReadyReq (SP: CCR_ServicePrim) : bool is SP match CReadyReq endfunc
    function IsCReadyInd (SP: CCR_ServicePrim) : bool is SP match CReadyInd endfunc
    function IsCCommitReq (SP: CCR_ServicePrim) : bool is SP match CCommitReq endfunc
    function IsCCommitInd (SP: CCR_ServicePrim) : bool is SP match CCommitInd endfunc
    function IsCCommitRsp (SP: CCR_ServicePrim) : bool is SP match CCommitRsp endfunc
    function IsCCommitCnf (SP: CCR_ServicePrim) : bool is SP match CCommitCnf endfunc
    function IsCRollbackReq (SP: CCR_ServicePrim) : bool is SP match CRollbackReq endfunc
    function IsCRollbackInd (SP: CCR_ServicePrim) : bool is SP match CRollbackInd endfunc
    function IsCRollbackRsp (SP: CCR_ServicePrim) : bool is SP match CRollbackRsp endfunc
    function IsCRollbackCnf (SP: CCR_ServicePrim) : bool is SP match CRollbackCnf endfunc
    function IsCRecoverReq (SP: CCR_ServicePrim) : bool is SP match CRecoverReq endfunc
    function IsCRecoverInd (SP: CCR_ServicePrim) : bool is SP match CRecoverInd endfunc
    function IsCRecoverRsp (SP: CCR_ServicePrim) : bool is SP match CRecoverRsp endfunc
    function IsCRecoverCnf (SP: CCR_ServicePrim) : bool is SP match CRecoverCnf endfunc
/* IsCCRsp(sp) = (first_CCRsp le key(sp)) and (key(sp) le last_CCRsp); */

    function get_aaid (SP: CCR_ServicePrim) : atomic_action_identifier is
        select AAID in SP endfunc
    function get_aaimn (SP: CCR_ServicePrim) : AE_title is
        select AAIMN in SP endfunc
    function get_aais (SP: CCR_ServicePrim) : suffix is
        select AAIS in SP endfunc
    function get_brid (SP: CCR_ServicePrim) : branch_identifier is
        select BRID in SP endfunc
    function get_brisn (SP: CCR_ServicePrim) : AE_title is
        select BRISN in SP endfunc
    function get_bris (SP: CCR_ServicePrim) : suffix is
        select BRIS in SP endfunc
    function get_ud (SP: CCR_ServicePrim) : PDUqueue is
        select UD in SP endfunc
    function get_rsri (SP: CCR_ServicePrim) : recovery_state is
        select RSRI in SP endfunc
    function get_rsrc (SP: CCR_ServicePrim) : recovery_response is
        select RSRC in SP endfunc
endmod
========================================================================== *)


type CCR_ServicePrim_extra_operations is CCR_ServicePrim
opns
  CBeginReq : atomic_action_identifier, branch_identifier -> ServicePrim
  CCommit_CBeginReq :
    atomic_action_identifier, branch_identifier -> ServicePrim
  CPrepareReq : PDU     (*user_data*)  -> ServicePrim
  CCommitReq : PDU      (*user_data*)  -> ServicePrim
  CCommitReq : -> ServicePrim
  CCommitRsp : PDU      (*user_data*)  -> ServicePrim
  CCommitRsp : -> ServicePrim
  CRollbackReq : PDU    (*user_data*)  -> ServicePrim
  CRollbackReq : -> ServicePrim
  CRollbackRsp : PDU    (*user_data*)  -> ServicePrim
  CRollbackRsp : -> ServicePrim
  CRecoverReq :
    recovery_state,
    atomic_action_identifier,
    branch_identifier,
    PDU (*user_data*)  -> ServicePrim
  CRecoverReq : recovery_state, atomic_action_identifier, branch_identifier -> ServicePrim
  CRecoverRsp :
    recovery_response (*recovery_state*),
    atomic_action_identifier,
    branch_identifier,
    PDU (*user_data*)  -> ServicePrim
  CRecoverRsp :
    recovery_response (*recovery_state*),
    atomic_action_identifier,
    branch_identifier -> ServicePrim
eqns
    forall
      aaid : atomic_action_identifier,
      brid : branch_identifier,
      pdu : PDU (*user_data*),
      rsri : recovery_state,
      rsrc : recovery_response (*recovery_state*),
      sp : ServicePrim
  ofsort ServicePrim
    CBeginReq(aaid, brid) =
      CBeginReq( masters_name(aaid), suffix(aaid), superiors_name(brid), suffix(brid));
    CCommit_CBeginReq(aaid, brid) =
      CCommit_CBeginReq( masters_name(aaid), suffix(aaid), superiors_name(brid), suffix(brid));
    CPrepareReq(pdu) = CPrepareReq(queue(pdu));
    CCommitReq(pdu) = CCommitReq(queue(pdu));
    CCommitReq = CCommitReq(emptyPDU);
    CCommitRsp(pdu) = CCommitRsp(queue(pdu));
    CCommitRsp = CCommitRsp(emptyPDU);
    CRollbackReq(pdu) = CRollbackReq(queue(pdu));
    CRollbackReq = CRollbackReq(emptyPDU);
    CRollbackRsp(pdu) = CRollbackRsp(queue(pdu));
    CRollbackRsp = CRollbackRsp(emptyPDU);
    CRecoverReq(rsri, aaid, brid, pdu) = CRecoverReq(rsri, aaid, brid, queue(pdu));
    CRecoverReq(rsri, aaid, brid) = CRecoverReq(rsri, aaid, brid, emptyPDU);
    CRecoverRsp(rsrc, aaid, brid, pdu) = CRecoverRsp(rsrc, aaid, brid, queue(pdu));
    CRecoverRsp(rsrc, aaid, brid) = CRecoverRsp(rsrc, aaid, brid, emptyPDU);
  ofsort atomic_action_identifier
    IsCBeginReq(sp)
    or IsCBeginInd(sp)
    or IsCCommit_CBeginReq(sp)
    or IsCCommit_CBeginInd(sp) =>
      get_aaid(sp) = atomic_action_identifier(get_aaimn(sp), get_aais(sp))
  ofsort branch_identifier
    IsCBeginReq(sp)
    or IsCBeginInd(sp)
    or IsCCommit_CBeginReq(sp)
    or IsCCommit_CBeginInd(sp) =>
      get_brid(sp) = branch_identifier(get_brisn(sp), get_bris(sp))
endtype
(* ---------------------------------------------------------------------------
module CCR_ServicePrim_extra_operations is CCR_ServicePrim
```

```
    function CBeginReq (A: atomic_action_identifier, B: branch_identifier) : CCR_ServicePrim is
        CBeginReq (masters_name (A), suffix (A), superiors_name (B), suffix (B)) endfunc

    function CCommit_CBeginReq (A: atomic_action_identifier, B: branch_identifier) : CCR_ServicePrim is
        CCommit_CBeginReq (masters_name(A), suffix(A), superiors_name(B), suffix(B)) endfunc

    function CPrepareReq (P: PDU) : CCR_ServicePrim is
        CPrepareReq (queue(P)) endfunc

    function CCommitReq (P: PDU) : CCR_ServicePrim is
        CCommitReq (queue (P)) endfunc
    function CCommitReq : CCR_ServicePrim is
        CCommitReq (emptyPDU) endfunc
    function CCommitRsp (P: PDU) : CCR_ServicePrim is
        CCommitRsp (queue(P)) endfunc
    function CCommitRsp : CCR_ServicePrim is
        CCommitRsp (emptyPDU) endfunc
    function CRollbackReq (P: PDU) : CCR_ServicePrim is
        CRollbackReq (queue (P)) endfunc
    function CRollbackReq : CCR_ServicePrim is
        CRollbackReq (emptyPDU) endfunc
    function CRollbackRsp (P: PDU) : CCR_ServicePrim is
        CRollbackRsp (queue(P)) endfunc
    function CRollbackRsp : CCR_ServicePrim is
        CRollbackRsp (emptyPDU) endfunc
    function CRecoverReq (r: recovery_state, a: atomic_action_identifier,
        b: branch_identifier, p: PDU) : CCR_ServicePrim is
        CRecoverReq (r, a, b, queue(p)) endfunc
    function CRecoverReq (r: recovery_state, a: atomic_action_identifier,
        b: branch_identifier) : CCR_ServicePrim is
        CRecoverReq(r, a, b, emptyPDU) endfunc
    function CRecoverRsp(r: recovery_state, a: atomic_action_identifier,
        b: branch_identifier, p: PDU) : CCR_ServicePrim is
        CRecoverRsp(r, a, b, queue(p)) endfunc
    function CRecoverRsp(r: recovery_state, a: atomic_action_identifier,
        b: branch_identifier) : CCR_ServicePrim is
        CRecoverRsp(r, a, b, emptyPDU) endfunc

    function get_aaid (SP: CCR_ServicePrim) : CCR_ServicePrim is
        atomic_action_identifier (get_aaimn (SP), get_aais (SP)) endfunc
  ofsort branch_identifier
    function IsCBeginReq(sp)
    function or IsCBeginInd(sp)
    function or IsCCommit_CBeginReq(sp)
    function or IsCCommit_CBeginInd(sp) =>
    function get_brid (SP: CCR_ServicePrim) : CCR_ServicePrim is
        branch_identifier (get_brisn (SP), get_bris (SP))
endmod
========================================================================= *)

type ACSE_ServicePrim is CCR_ServicePrim_extra_operations
opns
  AAssocReq (*! constructor *) : a_associate_mode,
        application_context_name,
          (*calling*) address,
          (*called*) address,
        PDUqueue (*ACSE_user_information*),
        presentation_context_definition_list,
        QOS_ISO8326,
        session_requirements,
        initial_assignment_of_tokens -> ServicePrim
  AAssocInd (*! constructor *) : a_associate_mode,
        application_context_name,
          (*calling*) address,
          (*called*) address,
        PDUqueue (*ACSE_user_information*),
        QOS_ISO8326,
        initial_assignment_of_tokens -> ServicePrim
  AAssocRsp (*! constructor *) : application_context_name,
          (*responding*) address,
        PDUqueue (*ACSE_user_information*),
        a_associate_result,
        QOS_ISO8326,
        initial_assignment_of_tokens -> ServicePrim
  AAssocCnf (*! constructor *) : application_context_name,
          (*responding*) address,
        PDUqueue (*ACSE_user_information*),
        a_associate_result,
        a_associate_result_source,
        a_associate_diagnostic_Opt,
        QOS_ISO8326,
        initial_assignment_of_tokens -> ServicePrim
  AReleaseReq (*! constructor *) : -> ServicePrim
  AReleaseInd (*! constructor *) : -> ServicePrim
  AReleaseRsp (*! constructor *) : a_release_result -> ServicePrim
  AReleaseCnf (*! constructor *) : a_release_result -> ServicePrim
  AAbortReq (*! constructor *) : PDUqueue (*ACSE_user_information*)      -> ServicePrim
  AAbortInd (*! constructor *) : PDUqueue (*ACSE_user_information*)      -> ServicePrim
  APAbortInd (*! constructor *) : -> ServicePrim

  AAssoc_req,
  AAssoc_ind,
  AAssoc_rsp,
  AAssoc_cnf,
  ARelease_req,
  ARelease_ind,
  ARelease_rsp,
  ARelease_cnf,
  AAbort_req,
  AAbort_ind,
  APAbort_ind,
  first_ACSEsp,
  last_ACSEsp : ->
      ServicePrimKey
  IsAAssocReq,
  IsAAssocInd,
  IsAAssocRsp,
  IsAAssocCnf,
  IsAReleaseReq,
  IsAReleaseInd,
  IsAReleaseRsp,
  IsAReleaseCnf,
  IsAAbortReq,
  IsAAbortInd,
  IsAPAbortInd,
  IsACSEsp :
    ServicePrim -> bool
  get_aam : ServicePrim -> a_associate_mode
  get_cllng : ServicePrim -> (*calling*) address
  get_clld : ServicePrim -> (*called*) address
  get_rspng : ServicePrim -> (*responding*) address
  get_aui : ServicePrim -> PDUqueue (*ACSE_user_information*)
  get_aar :
    ServicePrim -> a_associate_result
  get_aars : ServicePrim -> a_associate_result_source
  get_aadO : ServicePrim -> a_associate_diagnostic_Opt
  get_pcdl : ServicePrim -> presentation_context_definition_list
  get_sr : ServicePrim -> session_requirements
  get_iaot : ServicePrim -> initial_assignment_of_tokens
  get_arr : ServicePrim -> a_release_result
  get_qos : ServicePrim -> QOS_ISO8326
eqns
    forall
      aam : a_associate_mode,
      acn : application_context_name,
      cllng : (*calling*) address,
      clld :  (*called*) address,
      rspng : (*responding*) address,
      aui : PDUqueue (*ACSE_user_information*),
      aar : a_associate_result,
      aars : a_associate_result_source,
      aadO : a_associate_diagnostic_Opt,
      pcdl : presentation_context_definition_list,
      sr : session_requirements,
      iaot : initial_assignment_of_tokens,
      arr : a_release_result,
      qos : QOS_ISO8326,
      sp : ServicePrim
  ofsort ServicePrimKey
    first_ACSEsp = succ(last_CCRsp);
    AAssoc_req = first_ACSEsp;
    AAssoc_ind = succ(AAssoc_req);
    AAssoc_rsp = succ(AAssoc_ind);
    AAssoc_cnf = succ(AAssoc_rsp);
    ARelease_req = succ(AAssoc_cnf);
    ARelease_ind = succ(ARelease_req);
    ARelease_rsp = succ(ARelease_ind);
    ARelease_cnf = succ(ARelease_rsp);
    AAbort_req = succ(ARelease_cnf);
    AAbort_ind = succ(AAbort_req);
    APAbort_ind = succ(AAbort_ind);
    last_ACSEsp = APAbort_ind;
    key(AAssocReq(aam, acn, cllng, clld, aui, pcdl, qos, sr, iaot)) =
      AAssoc_req;
    key(AAssocInd(aam, acn, cllng, clld, aui, qos, iaot)) = AAssoc_ind;
    key(AAssocRsp(acn, rspng, aui, aar, qos, iaot)) = AAssoc_rsp;
    key(AAssocCnf(acn, rspng, aui, aar, aars, aad0, qos, iaot)) = AAssoc_cnf;
    key(AReleaseReq) = ARelease_req;
    key(AReleaseInd) = ARelease_ind;
    key(AReleaseRsp(arr)) = ARelease_rsp;
    key(AReleaseCnf(arr)) = ARelease_cnf;
    key(AAbortReq(aui)) = AAbort_req;
    key(AAbortInd(aui)) = AAbort_ind;
    key(APAbortInd) = APAbort_ind;
  ofsort bool
    IsAAssocReq(sp) = sp eq AAssoc_req;
    IsAAssocInd(sp) = sp eq AAssoc_ind;
    IsAAssocRsp(sp) = sp eq AAssoc_rsp;
    IsAAssocCnf(sp) = sp eq AAssoc_cnf;
    IsAReleaseReq(sp) = sp eq ARelease_req;
    IsAReleaseInd(sp) = sp eq ARelease_ind;
    IsAReleaseRsp(sp) = sp eq ARelease_rsp;
    IsAReleaseCnf(sp) = sp eq ARelease_cnf;
    IsAAbortReq(sp) = sp eq AAbort_req;
    IsAAbortInd(sp) = sp eq AAbort_ind;
    IsAPAbortInd(sp) = sp eq APAbort_ind;
    IsACSEsp(sp) = (first_ACSEsp le key(sp)) and (key(sp) le last_ACSEsp);
  ofsort a_associate_mode
    get_aam(AAssocReq(aam, acn, cllng, clld, aui, pcdl, qos, sr, iaot)) = aam;
    get_aam(AAssocInd(aam, acn, cllng, clld, aui, qos, iaot)) = aam;
  ofsort application_context_name
    get_acn(AAssocReq(aam, acn, cllng, clld, aui, pcdl, qos, sr, iaot)) = acn;
    get_acn(AAssocInd(aam, acn, cllng, clld, aui, qos, iaot)) = acn;
    get_acn(AAssocRsp(acn, rspng, aui, aar, qos, iaot)) = acn;
    get_acn(AAssocCnf(acn, rspng, aui, aar, aars, aad0, qos, iaot)) = acn;
  ofsort  (*calling*) address
    get_cllng(AAssocReq(aam, acn, cllng, clld, aui, pcdl, qos, sr, iaot)) =
      cllng;
    get_cllng(AAssocInd(aam, acn, cllng, clld, aui, qos, iaot)) = cllng;
  ofsort  (*called*) address
    get_clld(AAssocReq(aam, acn, cllng, clld, aui, pcdl, qos, sr, iaot)) = clld;
    get_clld(AAssocInd(aam, acn, cllng, clld, aui, qos, iaot)) = clld;
  ofsort  (*responding*) address
    get_rspng(AAssocRsp(acn, rspng, aui, aar, qos, iaot)) = rspng;
    get_rspng(AAssocCnf(acn, rspng, aui, aar, aars, aad0, qos, iaot)) = rspng;
  ofsort PDUqueue (*ACSE_user_information*)
    get_aui(
      AAssocReq(aam, acn, cllng, clld, aui, pcdl, qos, sr, iaot)) =
      aui;
    get_aui(AAssocInd(aam, acn, cllng, clld, aui, qos, iaot)) = aui;
    get_aui(AAssocRsp(acn, rspng, aui, aar, qos, iaot)) = aui;
    get_aui(AAssocCnf(acn, rspng, aui, aar, aars, aad0, qos, iaot)) = aui;
    get_aui(AAbortReq(aui)) = aui;
    get_aui(AAbortInd(aui)) = aui;
  ofsort a_associate_result
    get_aar(AAssocRsp(acn, rspng, aui, aar, qos, iaot)) = aar;
    get_aar(AAssocCnf(acn, rspng, aui, aar, aars, aadO, qos, iaot)) = aar;
  ofsort a_associate_result_source
    get_aars(AAssocCnf(acn, rspng, aui, aar, aars, aadO, qos, iaot)) = aars;
  ofsort a_associate_diagnostic_Opt
    get_aadO(AAssocCnf(acn, rspng, aui, aar, aars, aadO, qos, iaot)) = aadO;
  ofsort presentation_context_definition_list
    get_pcdl(AAssocReq(aam, acn, cllng, clld, aui, pcdl, qos, sr, iaot)) = pcdl;
  ofsort session_requirements
    get_sr(AAssocReq(aam, acn, cllng, clld, aui, pcdl, qos, sr, iaot)) = sr;
  ofsort initial_assignment_of_tokens
    get_iaot(AAssocReq(aam, acn, cllng, clld, aui, pcdl, qos, sr, iaot)) = iaot;
    get_iaot(AAssocInd(aam, acn, cllng, clld, aui, qos, iaot)) = iaot;
    get_iaot(AAssocRsp(acn, rspng, aui, aar, qos, iaot)) = iaot;
    get_iaot(AAssocCnf(acn, rspng, aui, aar, aars, aadO, qos, iaot)) = iaot;
  ofsort a_release_result
    get_arr(AReleaseRsp(arr)) = arr; get_arr(AReleaseCnf(arr)) = arr;
  ofsort QOS_ISO8326
    get_qos(AAssocReq(aam, acn, cllng, clld, aui, pcdl, qos, sr, iaot)) = qos;
    get_qos(AAssocInd(aam, acn, cllng, clld, aui, qos, iaot)) = qos;
    get_qos(AAssocRsp(acn, rspng, aui, aar, qos, iaot)) = qos;
    get_qos(AAssocCnf(acn, rspng, aui, aar, aars, aadO, qos, iaot)) = qos;
endtype
(* ---------------------------------------------------------------------------
module ACSE_ServicePrim is CCR_ServicePrim_extra_operations

    type ACSE_ServicePrim is
        AAssocReq (AAM: a_associate_mode,
                ACN: application_context_name,
                CLLNG: address, /*calling*/
                CLLD: address,  /*called*/
                AUI: PDUqueue,  /*ACSE_user_information*/
                PCDL: presentation_context_definition_list,
                QOS: QOS_ISO8326,
                SR: session_requirements,
                IAOT: initial_assignment_of_tokens)
    |   AAssocInd (AAM: a_associate_mode,
                ACN: application_context_name,
                CLLNG: address, /*calling*/
                CLLD: address,  /*called*/
                AUI: PDUqueue,  /*ACSE_user_information*/
                PCDL: presentation_context_definition_list,
                QOS: QOS_ISO8326,
                SR: session_requirements,
                IAOT: initial_assignment_of_tokens)
    |   AAssocRsp (ACN: application_context_name,
                RSPNG: address, /*responding*/
                AUI: PDUqueue,  /*ACSE_user_information*/
                AAR: a_associate_result,
                QOS: QOS_ISO8326,
                IAOT: initial_assignment_of_tokens)
    |   AAssocCnf (ACN: application_context_name,
                RSPNG: address, /*responding*/
                AUI: PDUqueue,  /*ACSE_user_information*/
                AAR: a_associate_result,
                AARS: a_associate_result_source,
                AADO: a_associate_diagnostic_Opt,
                QOS: QOS_ISO8326,
                IAOT: initial_assignment_of_tokens)
    |   AReleaseReq
    |   AReleaseInd
    |   AReleaseRsp (ARR: a_release_result)
    |   AReleaseCnf (ARR: a_release_result)
    |   AAbortReq (AUI: PDUqueue)       /*ACSE_user_information*/
    |   AAbortInd (AUI: PDUqueue)       /*ACSE_user_information*/
    |   APAbortInd
    endtype

/* no translation is needed for A*_{rsp|req|ind|cnf} */
/* no trasnlation is needed for key */

    function IsAAssocReq (SP: ACSE_ServicePrim) : bool is SP match AAssocReq endfunc
    function IsAAssocInd (SP: ACSE_ServicePrim) : bool is SP match AAssocInd endfunc
    function IsAAssocRsp (SP: ACSE_ServicePrim) : bool is SP match AAssocRsp endfunc
    function IsAAssocCnf (SP: ACSE_ServicePrim) : bool is SP match AAssocCnf endfunc
    function IsAReleaseReq (SP: ACSE_ServicePrim) : bool is SP match AReleaseReq endfunc
    function IsAReleaseInd (SP: ACSE_ServicePrim) : bool is SP match AReleaseInd endfunc
    function IsAReleaseRsp (SP: ACSE_ServicePrim) : bool is SP match AReleaseRsp endfunc
    function IsAReleaseCnf (SP: ACSE_ServicePrim) : bool is SP match AReleaseCnf endfunc
    function IsAAbortReq (SP: ACSE_ServicePrim) : bool is SP match AAbortReq endfunc
    function IsAAbortInd (SP: ACSE_ServicePrim) : bool is SP match AAbortInd endfunc
    function IsAPAbortInd (SP: ACSE_ServicePrim) : bool is SP match APAbortInd endfunc
/* IsACSEsp(sp) = (first_ACSEsp le key(sp)) and (key(sp) le last_ACSEsp); */

    function get_aam (SP: ACSE_ServicePrim) : a_associate_mode is
        select AAM in SP endfunc
    function get_acn (SP: ACSE_ServicePrim) : application_context_name is
        select ACN in SP endfunc
    function get_cllng (SP: ACSE_ServicePrim) : address is
        select CLLNG in SP endfunc
    function get_clld (SP: ACSE_ServicePrim) : address is
        select CLLD in SP endfunc
    function get_rspng (SP: ACSE_ServicePrim) : address is
        select RSPNG in SP endfunc
    function get_aui (SP: ACSE_ServicePrim) : PDUqueue is
        select AUI in SP endfunc
    function get_aar (SP: ACSE_ServicePrim) : a_associate_result is
        select AAR in SP endfunc
    function get_aars (SP: ACSE_ServicePrim) : a_associate_result_source is
        select AARS in SP endfunc
    function get_aadO (SP: ACSE_ServicePrim) : a_associate_diagnostic_Opt is
        select AADO in SP endfunc
    function get_pcdl(SP: ACSE_ServicePrim) : presentation_context_definition_list is
        select PCDL in SP endfunc
    function get_sr (SP: ACSE_ServicePrim) : session_requirements is
        select SR in SP endfunc
    function get_iaot (SP: ACSE_ServicePrim) : initial_assignment_of_tokens is
        select iaot in SP endfunc
    function get_arr (SP: ACSE_ServicePrim) : a_release_result is
```

```
        select ARR in SP endfunc
    function get_qos (SP: ACSE_ServicePrim) : QOS_ISO8326 is
        select QOS in SP endfunc
endmod
================================================================================ *)

type ACSE_ServicePrim_extra_operations is ACSE_ServicePrim
opns
  AAssocReq : a_associate_mode,
        application_context_name,
        (*calling*) address,
        (*called*) address,
        PDU (*ACSE_user_information*),
        presentation_context_definition_list,
        QOS_ISO8326,
        session_requirements,
        initial_assignment_of_tokens -> ServicePrim
  AAssocRsp : application_context_name,
        (*responding*) address,
        PDU (*ACSE_user_information*),
        a_associate_result,
        QOS_ISO8326,
        initial_assignment_of_tokens -> ServicePrim
  AAbortReq : PDU (*ACSE_user_information*)  -> ServicePrim
  AAbortReq : -> ServicePrim
eqns
    forall
      aam : a_associate_mode,
      acn : application_context_name,
      cllng : (*calling*) address,
      clld :  (*called*) address,
      rspng : (*responding*) address,
      pdu : PDU (*ACSE_user_information*),
      aar : a_associate_result,
      pcdl : presentation_context_definition_list,
      sr : session_requirements,
      iaot : initial_assignment_of_tokens,
      qos : QOS_ISO8326
  ofsort ServicePrim
    AAssocReq(aam, acn, cllng, clld, pdu, pcdl, qos, sr, iaot) =
      AAssocReq(aam, acn, cllng, clld, queue(pdu), pcdl, qos, sr, iaot);
    AAssocRsp(acn, rspng, pdu, aar, qos, iaot) =
      AAssocRsp(acn, rspng, queue(pdu), aar, qos, iaot);
    AAbortReq(pdu) = AAbortReq(queue(pdu));
    AAbortReq = AAbortReq(emptyPDU);
endtype
(* ----------------------------------------------------------------------------
module ACSE_ServicePrim_extra_operations is ACSE_ServicePrim
  function AAssocReq (aam: a_associate_mode,
        acn: application_context_name,
        cllng: address,
        clld: address,
        pdu: PDU,
        pcdl: presentation_context_definition_list,
        qos: QOS_ISO8326,
        sr: session_requirements,
        iaot: initial_assignment_of_tokens) : ACSE_ServicePrim is
    AAssocReq(aam, acn, cllng, clld, queue(pdu), pcdl, qos, sr, iaot) endfunc

  function AAssocRsp (acn: application_context_name,
        rspng: address,
        pdu: PDU,
        aar: a_associate_result,
        qos: QOS_ISO8326,
        iaot: initial_assignment_of_tokens) : ACSE_ServicePrim is
    AAssocRsp(acn, rspng, queue(pdu), aar, qos, iaot) endfunc

  function AAbortReq(pdu: PDU) : ACSE_ServicePrim is AAbortReq(queue(pdu)) endfunc

  function AAbortReq : ACSE_ServicePrim is AAbortReq(emptyPDU) endfunc
endmod
=========================================================================== *)

(*      The description of presentation service primitives is only
schematic: *)
(*      We assume 8822:DAM2:1992 for the user-data parameter on
PTokenGive
Req/Ind *)
(*      Parameters that are not used by TP are not shown; and the tokens
parameter on PToken Please/Give is not shown because it assumed to always
have the same value, i.e. "synchronize minor". *)
(*      Other PToken Please/Give are included in OtherPresentation SPs  *)

type P_ServicePrim is ACSE_ServicePrim_extra_operations
opns
  PTokenPleaseReq (*! constructor *) : -> ServicePrim
  PTokenPleaseInd (*! constructor *) : -> ServicePrim
  PTokenGiveReq (*! constructor *) : PDUqueue (*user_data*)  -> ServicePrim
  PTokenGiveInd (*! constructor *) : PDUqueue (*user_data*)  -> ServicePrim
  PDataReq (*! constructor *) : PDUqueue (*user_data*)  -> ServicePrim
  PDataInd (*! constructor *) : PDUqueue (*user_data*)  -> ServicePrim

  PTokenPlease_req,
  PTokenPlease_ind,
  PTokenGive_req,
  PTokenGive_ind,
  PData_req,
  PData_ind,
  PWithEmbeddedAPDU_ReqRsp,
  PWithEmbeddedAPDU_IndCnf,
  OtherPresentation_ReqRsp,
  OtherPresentation_IndCnf,
  first_Psp,
  last_Psp : ->
      ServicePrimKey
  IsPTokenPleaseReq,
  IsPTokenPleaseInd,
  IsPTokenGiveReq,
  IsPTokenGiveInd,
  IsPDataReq,
  IsPDataInd,
  IsPWithEmbeddedAPDUReqRsp,
  IsPWithEmbeddedAPDUIndCnf,
  IsOtherPresentationReqRsp,
  IsOtherPresentationIndCnf,
  assigns_token_here,
  assigns_token_there,
  IsPsp :
    ServicePrim -> bool
eqns
    forall ud : PDUqueue (*user_data*), sp : ServicePrim
  ofsort ServicePrimKey
    first_Psp = succ(last_ACSEsp);
    PTokenPlease_req = first_Psp;
    PTokenPlease_ind = succ(PTokenPlease_req);
    PTokenGive_req = succ(PTokenPlease_ind);
    PTokenGive_ind = succ(PTokenGive_req);
    PData_req = succ(PTokenGive_ind);
    PData_ind = succ(PData_req);
    PWithEmbeddedAPDU_ReqRsp = succ(PData_ind);
    PWithEmbeddedAPDU_IndCnf = succ(PWithEmbeddedAPDU_ReqRsp);
    OtherPresentation_ReqRsp = succ(PWithEmbeddedAPDU_IndCnf);
    OtherPresentation_IndCnf = succ(OtherPresentation_ReqRsp);
    last_Psp = OtherPresentation_IndCnf;
    key(PTokenPleaseReq) = PTokenPlease_req;
    key(PTokenPleaseInd) = PTokenPlease_ind;
    key(PTokenGiveReq(ud)) = PTokenGive_req;
    key(PTokenGiveInd(ud)) = PTokenGive_ind;
    key(PDataReq(ud)) = PData_req;
    key(PDataInd(ud)) = PData_ind;
  ofsort bool
    IsPTokenPleaseReq(sp) = sp eq PTokenPlease_req;
    IsPTokenPleaseInd(sp) = sp eq PTokenPlease_ind;
    IsPTokenGiveReq(sp) = sp eq PTokenGive_req;
    IsPTokenGiveInd(sp) = sp eq PTokenGive_ind;
    IsPDataReq(sp) = sp eq PData_req;
    IsPDataInd(sp) = sp eq PData_ind;
    IsPWithEmbeddedAPDUReqRsp(sp) = sp eq PWithEmbeddedAPDU_ReqRsp;
    IsPWithEmbeddedAPDUIndCnf(sp) = sp eq PWithEmbeddedAPDU_IndCnf;
    IsOtherPresentationReqRsp(sp) = sp eq OtherPresentation_ReqRsp;
    IsOtherPresentationIndCnf(sp) = sp eq OtherPresentation_IndCnf;
    IsPsp(sp) = (first_Psp le key(sp)) and (key(sp) le last_Psp);
  ofsort PDUqueue (*user_data*)
    get_ud( PTokenGiveReq(ud)) = ud;
    get_ud(PTokenGiveInd(ud)) = ud;
    get_ud(PDataReq(ud)) = ud;
    get_ud(PDataInd(ud)) = ud;
endtype
(* -------------------------------------------------------------------------
module P_ServicePrim is ACSE_ServicePrim_extra_operations

    type P_ServicePrim is
```

```
        PTokenPleaseReq
    |   PTokenPleaseInd
    |   PTokenGiveReq (UD: PDUqueue)    /*user_data*/
    |   PTokenGiveInd (UD: PDUqueue)    /*user_data*/
    |   PDataReq (UD: PDUqueue) /*user_data*/
    |   PDataInd (UD: PDUqueue) /*user_data*/
    endtype

    function IsPTokenPleaseReq (SP: P_ServicePrim) : bool is SP match PTokenPleaseReq endfunc
    function IsPTokenPleaseInd (SP: P_ServicePrim) : bool is SP match PTokenPleaseInd endfunc
    function IsPTokenGiveReq (SP: P_ServicePrim) : bool is SP match PTokenGiveReq (...) endfunc
    function IsPTokenGiveInd (SP: P_ServicePrim) : bool is SP match PTokenGiveInd (...) endfunc
    function IsPDataReq (SP: P_ServicePrim) : bool is SP match PDataReq (...) endfunc
    function IsPDataInd (SP: P_ServicePrim) : bool is SP match PDataInd (...) endfunc

/* this function are useless
    IsPWithEmbeddedAPDUReqRsp(sp) = sp eq PWithEmbeddedAPDU_ReqRsp;
    IsPWithEmbeddedAPDUIndCnf(sp) = sp eq PWithEmbeddedAPDU_IndCnf;
    IsOtherPresentationReqRsp(sp) = sp eq OtherPresentation_ReqRsp;
    IsOtherPresentationIndCnf(sp) = sp eq OtherPresentation_IndCnf;
    IsPsp(sp) = (first_Psp le key(sp)) and (key(sp) le last_Psp);
*/

    function get_ud (SP: P_ServicePrim) : PDUqueue is select UD in SP endfunc
endmod
============================================================================ *)

type P_ServicePrim_extra_operations is P_ServicePrim
opns
  PTokenGiveReq : PDU   (*user_data*)  -> ServicePrim
  PTokenGiveReq : -> ServicePrim
eqns
    forall pdu : PDU    (*user_data*)
  ofsort ServicePrim
    PTokenGiveReq(pdu) = PTokenGiveReq(queue(pdu));
    PTokenGiveReq = PTokenGiveReq(emptyPDU);
endtype
(* ---------------------------------------------------------------------------
module P_ServicePrim_extra_operations is P_ServicePrim
    function PTokenGiveReq (pdu: PDU) : P_ServicePrim is PTokenGiveReq(queue(pdu)) endfunc
    function PTokenGiveReq : P_ServicePrim is PTokenGiveReq(emptyPDU) endfunc
endmod
============================================================================ *)

type SpecialKeys is P_ServicePrim_extra_operations
sorts LookUpKey, LookUpKeyList
opns
  AFAbort_indi,
  AFAbort_ind_commitRI (*! constructor *),
  AFAbort_requ,
  AFAbort_req_user (*! constructor *),
  AFAbort_req_commitRI (*! constructor *),
  AFBeginDialogue_requ,
  AFBeginDialogue_resp,
  AFDefer_requ,
  AFEndDialogue_requ,
  AFHandshake_requ,
  AFHandShakeAndGrantControl_requ (*! constructor *),
  AFPrepare_requ,
  AFPrepare_req_true (*! constructor *),
  AFPrepare_indi,
  AFUError_resp,
  AFTokenPlease_requ,
  CAFDetach_clean_up (*! constructor *),
  CAFDetach_not_used (*! constructor *),
  CAFPlease_requ,
  CBegin_conf,
  CBegin_indi,
  CBegin_resp,
  CCommit_CBegin_indi,
  CCommit_CBegin_requ,
  CCommit_requ,
  CCommit_conf,
  CReady_indi,
  CRecover_req_commit (*! constructor *),
  RecoverInd (*! constructor *),
  CRecover_resp,
  CRecover_rsp_retry_later (*! constructor *),
  CRecover_cnf_done (*! constructor *),
  CommitCnf (*! constructor *),
  CommitInd (*! constructor *),
  CommitReq (*! constructor *),
  DeferEndDialogue (*! constructor *),
  DeferGrantControl (*! constructor *),
  RollbackCnf (*! constructor *),
  RollbackInd (*! constructor *),
  RollbackReq (*! constructor *),
  RollbackRsp (*! constructor *),
  TPBeginDialogue_indi,
  TPBeginTransaction_requ,
  TPBeginTransaction_req_purging (*! constructor *),
  TPCommit_indi,
  TPCommit_requ,
  TPRollback_indi,
  TPUAbort_indi,
  TPUAbort_requ,
  Abort (*! constructor *) : -> LookUpKey
  look_up (*! constructor *) : ServicePrimKey -> LookUpKey
  _eq_ : ServicePrim, LookUpKey -> bool
  nilKey (*! constructor *) : -> LookUpKeyList
  _+_ (*! constructor *) : LookUPKeyList, LookUpKey -> LookUpKeyList
  _IsIn_ : ServicePrim, LookUpKeyList -> bool
  commitSPs : -> LookUpKeyList
eqns
    forall
      sp : ServicePrim, spk : ServicePrimKey, l : LookUpKeyList, x : LookUpKey
  ofsort bool
    sp eq look_up(spk) = sp eq spk;
    sp IsIn nilKey = false;
    sp IsIn (l + x) = (sp eq x) or (sp IsIn l)
  ofsort LookUpKey
    AFAbort_indi = look_up(AFAbort_ind);
    AFAbort_requ = look_up(AFAbort_req);
    AFBeginDialogue_requ = look_up(AFBeginDialogue_req);
    AFBeginDialogue_resp = look_up(AFBeginDialogue_rsp);
    AFDefer_requ = look_up(AFDefer_req);
    AFEndDialogue_requ = look_up(AFEndDialogue_req);
    AFHandshake_requ = look_up(AFHandshake_req);
    AFPrepare_requ = look_up(AFPrepare_req);
    AFPrepare_indi = look_up(AFPrepare_ind);
    AFUError_resp = look_up(AFUError_rsp);
    AFTokenPlease_requ = look_up(AFTokenPlease_req);
    CAFPlease_requ = look_up(CAFPlease_req);
    CBegin_conf = look_up(CBegin_cnf);
    CBegin_indi = look_up(CBegin_ind);
    CBegin_resp = look_up(CBegin_rsp);
    CCommit_CBegin_indi = look_up(CCommit_CBegin_ind);
    CCommit_CBegin_requ = look_up(CCommit_CBegin_req);
    CCommit_requ = look_up(CCommit_req);
    CCommit_conf = look_up(CCommit_cnf);
    CReady_indi = look_up(CReady_ind);
    CRecover_resp = look_up(CRecover_rsp);
    TPBeginDialogue_indi = look_up(TPBeginDialogue_ind);
    TPBeginTransaction_requ = look_up(TPBeginTransaction_req);
    TPCommit_indi = look_up(TPCommit_ind);
    TPCommit_requ = look_up(TPCommit_req);
    TPRollback_indi = look_up(TPRollback_ind);
    TPUAbort_indi = look_up(TPUAbort_ind);
    TPUAbort_requ = look_up(TPUAbort_req);
  ofsort LookUpKeyList
    commitSPs =
      nilKey +
      AFPrepare_requ +
      CBegin_conf +
      CBegin_indi +
      CBegin_resp +
      CCommit_CBegin_indi +
      CCommit_CBegin_requ +
      CReady_indi +
      CommitCnf +
      CommitInd +
      CommitReq +
      RollbackCnf +
      RollbackInd +
      RollbackReq +
      RollbackRsp +
      TPCommit_indi +
      TPCommit_requ +
      TPRollback_indi;
endtype
(* ------------------------------------------------------------------------
module SpecialKeys is P_ServicePrim_extra_operations
```

```
/* LookUpKey */
    type LookUpKey is
        look_up (SPK: ServicePrimkey)
    |   AFAbort_ind_commitRI
    |   AFAbort_req_user
    |   AFAbort_req_commitRI
    |   AFHandShakeAndGrantControl_requ
    |   AFPrepare_req_true
    |   CAFDetach_clean_up
    |   CAFDetach_not_used
    |   CRecover_req_commit
    |   RecoverInd
    |   CRecover_rsp_retry_later
    |   CRecover_cnf_done
    |   CommitCnf
    |   CommitInd
    |   CommitReq
    |   DeferEndDialogue
    |   DeferGrantControl
    |   RollbackCnf
    |   RollbackInd
    |   RollbackReq
    |   RollbackRsp
    |   TPBeginTransaction_req_purging
    |   Abort
    endtype

/* LookUpKeyList */
    type LookUpKeyList is
        nilKey
    |   _+_ (L: LookUPKeyList, K: LookUpKey)
    endtype

  function AFAbort_indi : LookUpKey is look_up (AFAbort_ind) endfunc
  function AFAbort_requ : LookUpKey is look_up (AFAbort_req) endfunc
  function AFBeginDialogue_requ : LookUpKey is look_up (AFBeginDialogue_req) endfunc
  function AFBeginDialogue_resp : LookUpKey is look_up (AFBeginDialogue_rsp) endfunc
  function AFDefer_requ : LookUpKey is look_up (AFDefer_req) endfunc
  function AFEndDialogue_requ : LookUpKey is look_up (AFEndDialogue_req) endfunc
  function AFHandshake_requ : LookUpKey is look_up (AFHandshake_req) endfunc
  function AFPrepare_requ : LookUpKey is look_up (AFPrepare_req) endfunc
  function AFPrepare_indi : LookUpKey is look_up (AFPrepare_ind) endfunc
  function AFUError_resp : LookUpKey is look_up (AFUError_rsp) endfunc
  function AFTokenPlease_requ : LookUpKey is look_up (AFTokenPlease_req) endfunc
  function CAFPlease_requ : LookUpKey is look_up (CAFPlease_req) endfunc
  function CBegin_conf : LookUpKey is look_up (CBegin_cnf) endfunc
  function CBegin_indi : LookUpKey is look_up (CBegin_ind) endfunc
  function CBegin_resp : LookUpKey is look_up (CBegin_rsp) endfunc
  function CCommit_CBegin_indi : LookUpKey is look_up (CCommit_CBegin_ind) endfunc
  function CCommit_CBegin_requ : LookUpKey is look_up (CCommit_CBegin_req) endfunc
  function CCommit_requ : LookUpKey is look_up (CCommit_req) endfunc
  function CCommit_conf : LookUpKey is look_up (CCommit_cnf) endfunc
  function CReady_indi : LookUpKey is look_up (CReady_ind) endfunc
  function CRecover_resp : LookUpKey is look_up (CRecover_rsp) endfunc
  function TPBeginDialogue_indi : LookUpKey is look_up (TPBeginDialogue_ind) endfunc
  function TPBeginTransaction_requ : LookUpKey is look_up (TPBeginTransaction_req) endfunc
  function TPCommit_indi : LookUpKey is look_up (TPCommit_ind) endfunc
  function TPCommit_requ : LookUpKey is look_up (TPCommit_req) endfunc
  function TPRollback_indi : LookUpKey is look_up (TPRollback_ind) endfunc
  function TPUAbort_indi : LookUpKey is look_up (TPUAbort_ind) endfunc
  function TPUAbort_requ : LookUpKey is look_up (TPUAbort_req) endfunc

  function _eq_ (SP: ServicePrim, K: LookUpKey) : bool is
/* equations shifted from ServicePrim type */
        case LK in
            AFAbort_ind_commitRI -> IsAFAbortInd (SP) and is_map_commitRI (SP)
        |   AFAbort_req_user -> IsAFAbortReq_user (SP)
        |   AFAbort_req_commitRI -> IsAFAbortReq (SP) and is_map_commitRI (SP)
        |   AFPrepare_req_true -> IsAFPrepareReq (SP) and
                 not(is_absent(get_dp0 (SP))) andget_bool(get_dp0 (SP))
        |   Abort -> IsAFAbortReq (SP) or IsAFAbortInd (SP) or IsAAbortReq (SP)
                  or IsAAbortInd (SP) or IsAPAbortInd (SP)
        |   CAFDetach_clean_up
                -> IsCAFDetachReq (SP) and is_clean_up (get_dt (SP))
        |   CAFDetach_not_used
                -> IsCAFDetachReq (SP) and is_not_used (get_dt (SP))
        |   CRecover_req_commit
                -> IsCRecoverReq (SP) and is_commit (get_rsri (SP))
        |   RecoverInd -> IsCRecoverInd (SP) or
                IsAFRecoverInd (SP) or IsCAFRecoverInd (SP)
        |   CRecover_rsp_retry_later -> IsCRecoverRsp_retry_later (SP)
        |   CRecover_cnf_done -> IsCRecoverCnf (SP) and is_done (get_rsrc (SP))
        |   CommitCnf -> IsCommitCnf (SP)
        |   CommitInd -> IsCommitInd (SP)
        |   CommitReq -> IsCommitReq (SP)
        |   DeferEndDialogue -> IsDeferEndDialogue (SP)
        |   DeferGrantControl -> IsDeferGrantControl (SP)
        |   RollbackCnf -> IsRollbackCnf (SP)
        |   RollbackInd -> IsRollbackInd (SP)
        |   RollbackReq -> IsRollbackReq (SP)
        |   RollbackRsp -> IsRollbackRsp (SP)
        |   look_up (SPK:=SPK : ServicePrimkey) -> SP eq SPK
  endfunc

  function _IsIn_ (SP: ServicePrim, L: LookUpKeyList) : bool is
        case L in
            nilKey -> false
        |   (L1: LookUpKeyList) + (K: LookUpKey) -> (SP eq K) or (SP IsIn L1)
        endcase
  endfunc

  function commitSPs : LookUpKeyList is
      nilKey +
      AFPrepare_requ +
      CBegin_conf +
      CBegin_indi +
      CBegin_resp +
      CCommit_CBegin_indi +
      CCommit_CBegin_requ +
      CReady_indi +
      CommitCnf +
      CommitInd +
      CommitReq +
      RollbackCnf +
      RollbackInd +
      RollbackReq +
      RollbackRsp +
      TPCommit_indi +
      TPCommit_requ +
      TPRollback_indi
  endfunc

endmod
========================================================================== *)

type ServicePrim is make_mapping_parameter_from_ServicePrimKey
opns
  IsReq,
  IsInd,
  IsRsp,
  IsCnf,
  IsReqRsp,
  IsIndCnf,
  IsTPBeginDialogueRsp_rejected,
  IsTPBeginDialogueRsp_accepted,
  IsTPBeginDialogueCnf_rejected,
  IsTPBeginDialogueCnf_accepted,
  IsAFAbortReq_user,
  IsCRecoverRsp_retry_later,
  IsCAFDetachReq_clean_up,
  IsCommitReq,
  IsCommitInd,
  IsCommitCnf,
  IsRollbackReq,
  IsRollbackInd,
  IsRollbackRsp,
  IsRollbackCnf : ServicePrim -> bool
  CBeginReq_CRecoverRspUnknown :
    ServicePrimKey, atomic_action_identifier, branch_identifier -> ServicePrim
  CRecover_AFRecover_Req :
    recovery_state,
    recovery_context_handle_Opt,
    atomic_action_identifier,
    branch_identifier -> ServicePrim
  absentSP (*! constructor *) : -> ServicePrim
  is_absent : ServicePrim -> bool
  is_user, is_provider : ServicePrim -> bool
  get_addr : ServicePrim -> address
eqns
    forall sp : ServicePrim
  ofsort bool
```

```
IsTPBeginDialogueRsp_rejected(sp) = IsTPBeginDialogueRsp(sp) and is_rejected(get_bdr(sp));
IsTPBeginDialogueRsp_accepted(sp) = IsTPBeginDialogueRsp(sp) and is_accepted(get_bdr(sp));
IsTPBeginDialogueCnf_rejected(sp) = IsTPBeginDialogueCnf(sp) and is_rejected(get_bdr(sp));
IsTPBeginDialogueCnf_accepted(sp) = IsTPBeginDialogueCnf(sp) and is_accepted(get_bdr(sp));
IsAFAbortReq_user(sp) = IsAFAbortReq(sp) and is_user(sp);
IsCRecoverRsp_retry_later(sp) = IsCRecoverRsp(sp) and is_retry_later(get_rsrc(sp));
IsCommitCnf(sp) =
  IsCCommitCnf(sp) or
  (IsCRecoverCnf(sp) and is_done(get_rsrc(sp))) or
  ( ( IsAFHeuristicReportInd(sp) or
      IsAFAbortInd(sp) or
      IsAFAbortAndHeuristicReportInd(sp)
    ) and
    is_map_CommitRC(sp)
  ) or
  (IsAFHeuristicReportInd(sp) and is_map_recover_doneRC(sp));
IsCommitInd(sp) =
  IsCCommitInd(sp) or
  IsCCommit_CBeginInd(sp) or
  (IsAFAbortInd(sp) and is_user(sp) and is_map_commitRI(sp)) or
  ((IsCRecoverInd(sp) or IsAFRecoverInd(sp)) and is_commit(get_rsri(sp)));
IsCommitReq(sp) =
  IsCCommitReq(sp) or
  IsCCommit_CBeginReq(sp) or
  (IsAFAbortReq(sp) and is_user(sp) and is_map_commitRI(sp)) or
  ((IsCRecoverReq(sp) or IsAFRecoverReq(sp)) and is_commit(get_rsri(sp)));
IsRollbackCnf(sp) =
  IsCRollbackCnf(sp) or
  ( ( IsAFHeuristicReportInd(sp) or
      IsAFAbortInd(sp) or
      IsAFAbortAndHeuristicReportInd(sp) or
      IsAFBeginDialogueCnf(sp)
    ) and
    is_map_rollbackRC(sp)
  );
IsRollbackInd(sp) =
  IsCRollbackInd(sp) or
  ( ( IsAFHeuristicReportInd(sp) or
      IsAFAbortInd(sp) or
      IsAFAbortAndHeuristicReportInd(sp) or
      IsAFBeginDialogueCnf(sp)
    ) and
    is_map_rollbackRI(sp)
  );
IsRollbackReq(sp) =
  IsCRollbackReq(sp) or
  ( ( IsAFHeuristicReportReq(sp) or
      IsAFAbortReq(sp) or
      IsAFAbortAndHeuristicReportReq(sp)
    ) and
    is_map_rollbackRI(sp)
  );
IsRollbackRsp(sp) =
  IsCRollbackRsp(sp) or
```

```
      ( ( IsAFHeuristicReportReq(sp) or
          IsAFAbortReq(sp) or
          IsAFAbortAndHeuristicReportReq(sp)
        ) and
        is_map_rollbackRI(sp)
      );
    sp eq AFAbort_ind_commitRI = IsAFAbortInd(sp) and is_map_commitRI(sp);
    sp eq AFAbort_req_user = IsAFAbortReq_user(sp);
    sp eq AFAbort_req_commitRI = IsAFAbortReq(sp) and is_map_commitRI(sp);
    sp eq AFPrepare_req_true =
      IsAFPrepareReq(sp) and
      not(is_absent(get_dp0(sp))) and
      get_bool(get_dp0(sp));
    sp eq Abort =
      IsAFAbortReq(sp) or
      IsAFAbortInd(sp) or
      IsAAbortReq(sp) or
      IsAAbortInd(sp) or
      IsAPAbortInd(sp);
    sp eq CAFDetach_clean_up = IsCAFDetachReq(sp) and is_clean_up(get_dt(sp));
    sp eq CAFDetach_not_used = IsCAFDetachReq(sp) and is_not_used(get_dt(sp));
    sp eq CRecover_req_commit = IsCRecoverReq(sp) and is_commit(get_rsri(sp));
    sp eq RecoverInd =
      IsCRecoverInd(sp) or IsAFRecoverInd(sp) or IsCAFRecoverInd(sp);
    sp eq CRecover_rsp_retry_later = IsCRecoverRsp_retry_later(sp);
    sp eq CRecover_cnf_done = IsCRecoverCnf(sp) and is_done(get_rsrc(sp));
    sp eq CommitCnf = IsCommitCnf(sp);
    sp eq CommitInd = IsCommitInd(sp);
    sp eq CommitReq = IsCommitReq(sp);
    sp eq DeferEndDialogue = IsDeferEndDialogue(sp);
    sp eq DeferGrantControl = IsDeferGrantControl(sp);
    sp eq RollbackCnf = IsRollbackCnf(sp);
    sp eq RollbackInd = IsRollbackInd(sp);
    sp eq RollbackReq = IsRollbackReq(sp);
    sp eq RollbackRsp = IsRollbackRsp(sp);
    IsReq(sp) =
      IsTPDataReq(sp) or
      IsTPBeginDialogueReq(sp) or
      IsTPEndDialogueReq(sp) or
      IsTPUErrorReq(sp) or
      IsTPUAbortReq(sp) or
      IsTPGrantControlReq(sp) or
      IsTPRequestControlReq(sp) or
      IsTPHandshakeReq(sp) or
      IsTPHandshakeAndGrantControlReq(sp) or
      IsTPBeginTransactionReq(sp) or
      IsTPDeferredEndDialogueReq(sp) or
      IsTPDeferredGrantControlReq(sp) or
      IsTPCommitReq(sp) or
      IsTPDoneReq(sp) or
      IsTPPrepareReq(sp) or
      IsTPRollbackReq(sp) or
      IsAFBeginDialogueReq(sp) or
      IsAFBidReq(sp) or
      IsAFEndDialogueReq(sp) or
      IsAFUErrorReq(sp) or
      IsAFAbortReq(sp) or
      IsAFGrantControlReq(sp) or
      IsAFRequestControlReq(sp) or
      IsAFHandshakeReq(sp) or
      IsAFHandshakeAndGrantControlReq(sp) or
      IsAFDeferReq(sp) or
      IsAFPrepareReq(sp) or
      IsAFHeuristicReportReq(sp) or
      IsAFAbortAndHeuristicReportReq(sp) or
      IsAFTokenPleaseReq(sp) or
      IsAFTokenGiveReq(sp) or
      IsAFRecoverReq(sp) or
      IsCAFPleaseReq(sp) or
      IsCAFDetachReq(sp) or
      IsSAFDetachAssociationReq(sp) or
      IsCCommit_CBeginReq(sp) or
      IsCBeginReq(sp) or
      IsCPrepareReq(sp) or
      IsCReadyReq(sp) or
      IsCCommitReq(sp) or
      IsCRollbackReq(sp) or
      IsCRecoverReq(sp) or
      IsAAssocReq(sp) or
      IsAReleaseReq(sp) or
      IsAAbortReq(sp) or
      IsUaseReq(sp) or
      IsPDataReq(sp) or
      IsPTokenPleaseReq(sp) or
      IsPTokenGiveReq(sp);
    IsRsp(sp) =
      IsTPBeginDialogueRsp(sp) or
      IsTPEndDialogueRsp(sp) or
      IsTPHandshakeRsp(sp) or
      IsTPHandshakeAndGrantControlRsp(sp) or
      IsAFBeginDialogueRsp(sp) or
      IsAFBidRsp(sp) or
      IsAFEndDialogueRsp(sp) or
      IsAFUErrorRsp(sp) or
      IsAFHandshakeRsp(sp) or
      IsAFHandshakeAndGrantControlRsp(sp) or
      IsCBeginRsp(sp) or
      IsCCommitRsp(sp) or
      IsCRollbackRsp(sp) or
      IsCRecoverRsp(sp) or
      IsAAssocRsp(sp) or
      IsAReleaseRsp(sp);
    IsInd(sp) =
      IsTPDataInd(sp) or
      IsTPBeginDialogueInd(sp) or
      IsTPEndDialogueInd(sp) or
      IsTPUErrorInd(sp) or
      IsTPUAbortInd(sp) or
      IsTPPAbortInd(sp) or
      IsTPGrantControlInd(sp) or
      IsTPRequestControlInd(sp) or
      IsTPHandshakeInd(sp) or
      IsTPHandshakeAndGrantControlInd(sp) or
      IsTPHeuristicReportInd(sp) or
      IsTPBeginTransactionInd(sp) or
      IsTPDeferredEndDialogueInd(sp) or
      IsTPDeferredGrantControlInd(sp) or
      IsTPCommitInd(sp) or
      IsTPCommitCompleteInd(sp) or
      IsTPPrepareInd(sp) or
      IsTPReadyInd(sp) or
      IsTPRollbackInd(sp) or
      IsTPRollbackCompleteInd(sp) or
      IsAFBeginDialogueInd(sp) or
      IsAFBidInd(sp) or
      IsAFEndDialogueInd(sp) or
      IsAFUErrorInd(sp) or
      IsAFAbortInd(sp) or
      IsAFGrantControlInd(sp) or
      IsAFRequestControlInd(sp) or
      IsAFHandshakeInd(sp) or
      IsAFHandshakeAndGrantControlInd(sp) or
      IsAFDeferInd(sp) or
      IsAFPrepareInd(sp) or
      IsAFHeuristicReportInd(sp) or
      IsAFAbortAndHeuristicReportInd(sp) or
      IsAFTokenPleaseInd(sp) or
      IsAFTokenGiveInd(sp) or
      IsAFRecoverInd(sp) or
      IsCAFGiveInd(sp) or
      IsCAFRecoverInd(sp) or
      IsSAFAssociationLostInd(sp) or
      IsCCommit_CBeginInd(sp) or
      IsCBeginInd(sp) or
      IsCPrepareInd(sp) or
      IsCReadyInd(sp) or
      IsCCommitInd(sp) or
      IsCRollbackInd(sp) or
      IsCRecoverInd(sp) or
      IsAAssocInd(sp) or
      IsAReleaseInd(sp) or
      IsAAbortInd(sp) or
      IsAPAbortInd(sp) or
      IsUaseInd(sp) or
      IsPDataInd(sp) or
      IsPTokenPleaseInd(sp) or
      IsPTokenGiveInd(sp);
    IsCnf(sp) =
      IsTPBeginDialogueCnf(sp) or
      IsTPEndDialogueCnf(sp) or
      IsTPHandshakeCnf(sp) or
      IsTPHandshakeAndGrantControlCnf(sp) or
      IsAFBeginDialogueCnf(sp) or
      IsAFBidCnf(sp) or
      IsAFEndDialogueCnf(sp) or
      IsAFUErrorCnf(sp) or
      IsAFHandshakeCnf(sp) or
      IsAFHandshakeAndGrantControlCnf(sp) or
      IsCBeginCnf(sp) or
      IsCCommitCnf(sp) or
      IsCRollbackCnf(sp) or
      IsCRecoverCnf(sp) or
      IsAAssocCnf(sp) or
      IsAReleaseCnf(sp);
    IsReqRsp(sp) =
      IsReq(sp) or
      IsRsp(sp) or
      IsPWithEmbeddedAPDUReqRsp(sp) or
      IsOtherPresentationReqRsp(sp);
    IsIndCnf(sp) =
      IsInd(sp) or
      IsCnf(sp) or
      IsPWithEmbeddedAPDUIndCnf(sp) or
      IsOtherPresentationIndCnf(sp);
    get_type(sp) = user => is_user(sp) = true;
    get_type(sp) = provider => is_user(sp) = false;
    is_provider(sp) = not(is_user(sp));
    is_absent(sp) = sp eq succ(last_Psp);
  ofsort tp_abort_type
    IsTPUAbortReq(sp) or IsTPUAbortInd(sp) => get_type(sp) = user;
    IsTPPAbortInd(sp) => get_type(sp) = provider;
  ofsort address
    IsTPBeginDialogueReq(sp) =>
      get_addr(sp) = address(Opt(get_rapt(sp)), get_rapiO(sp), get_raeqO(sp), get_raeiO(sp));
    IsTPBeginDialogueInd(sp) =>
      get_addr(sp) = address(get_iaptO(sp), get_iapiO(sp), get_iaeqO(sp), get_iaeiO(sp));
  ofsort ServicePrim
      forall
        aet : AE_title,
        aaid : atomic_action_identifier,
        brid : branch_identifier,
        rsri : recovery_state,
        rch : recovery_context_handle,
        spk : ServicePrimKey
    spk eq CBegin_req =>
        CBeginReq_CRecoverRspUnknown(spk, aaid, brid) = CBeginReq(aaid, brid);
    spk eq CRecover_rsp =>
        CBeginReq_CRecoverRspUnknown(spk, aaid, brid) = CRecoverRsp(unknown, aaid, brid);
    CRecover_AFRecover_Req(rsri, rch_absent, aaid, brid) = CRecoverReq(rsri, aaid, brid);
    CRecover_AFRecover_Req(rsri, Opt(rch), aaid, brid) =
      AFRecoverReq(make_map(rsri), rch, aaid, brid)
  ofsort ServicePrimKey key(absentSP) = succ(last_Psp);
endtype
(* ----------------------------------------------------------------------------
module ServicePrim is make_mapping_parameter_from_ServicePrimKey
    type ServicePrim is
```

```
      TP (TP_SP: TP_ServicePrim)
   |  AF (AF_SP: AF_ServicePrim)
   |  CAF (CAF_SP: CAF_ServicePrim)
   |  SAF (SAF_SP: SAF_ServicePrim)
   |  CCR (CCR_SP: CCR_ServicePrim)
   |  ACSE (ACSE_SP: ACSE_ServicePrim)
   |  P (P_SP: P_ServicePrim)
   |  absentSP
   endtype

 function IsTPBeginDialogueRsp_rejected (SP: ServicePrim) : bool is
       IsTPBeginDialogueRsp (select TP_SP in SP)
       and is_rejected (get_bdr (select TP_SP in SP)) endfunc
 function IsTPBeginDialogueRsp_accepted (SP: ServicePrim) : bool is
       IsTPBeginDialogueRsp (select TP_SP in SP)
       and is_accepted (get_bdr (select TP_SP in SP)) endfunc
 function IsTPBeginDialogueCnf_rejected (SP: ServicePrim) : bool is
       IsTPBeginDialogueCnf (select TP_SP in SP)
       and is_rejected (get_bdr (select TP_SP in SP)) endfunc
 function IsTPBeginDialogueCnf_accepted (SP: ServicePrim) : bool is
       IsTPBeginDialogueCnf (select TP_SP in SP)
       and is_accepted (get_bdr (select TP_SP in SP)) endfunc
 function IsAFAbortReq_user (SP: ServicePrim) : bool is
       IsAFAbortReq (select AF_SP in SP)
       and is_user (select AF_SP in SP) endfunc
 function IsCRecoverRsp_retry_later (SP: ServicePrim) : bool is
       IsCRecoverRsp (select CCR_SP in SP)
       and is_retry_later (get_rsrc (select CCR_SP in SP)) endfunc
 function IsCommitCnf (SP: ServicePrim) : bool is
     IsCCommitCnf (SP) or
     (IsCRecoverCnf (SP) and is_done(get_rsrc (SP))) or
     ( ( IsAFHeuristicReportInd (SP) or
         IsAFAbortInd (SP) or
         IsAFAbortAndHeuristicReportInd (SP)
       ) and
       is_map_CommitRC (SP)
     ) or
     (IsAFHeuristicReportInd (SP) and is_map_recover_doneRC (SP))
 endfunc
 function IsCommitInd (SP: ServicePrim) : bool is
     IsCCommitInd (SP) or
     IsCCommit_CBeginInd (SP) or
     (IsAFAbortInd (SP) and is_user (SP) and is_map_commitRI (SP)) or
     ((IsCRecoverInd (SP) or IsAFRecoverInd (SP)) and is_commit(get_rsri (SP)))
 endfunc
 function IsCommitReq (SP: ServicePrim) : bool is
     IsCRollbackCnf (SP) or
     ( ( IsAFHeuristicReportInd (SP) or
         IsAFAbortInd (SP) or
         IsAFAbortAndHeuristicReportInd (SP) or
         IsAFBeginDialogueCnf (SP)
       ) and
       is_map_rollbackRC (SP)
     )
```

```
endfunc
function IsRollbackCnf (SP: ServicePrim) : bool is
    IsCRollbackCnf (SP) or
    ( ( IsAFHeuristicReportInd (SP) or
        IsAFAbortInd (SP) or
        IsAFAbortAndHeuristicReportInd (SP) or
        IsAFBeginDialogueCnf (SP)
      ) and
      is_map_rollbackRC (SP)
    ) endfunc
function IsRollbackInd (SP: ServicePrim) : bool is
    IsCRollbackInd (SP) or
    ( ( IsAFHeuristicReportInd (SP) or
        IsAFAbortInd (SP) or
        IsAFAbortAndHeuristicReportInd (SP) or
        IsAFBeginDialogueCnf (SP)
      ) and
      is_map_rollbackRI (SP)
    ) endfunc
function IsRollbackReq (SP: ServicePrim) : bool is
    IsCRollbackReq (SP) or
    ( ( IsAFHeuristicReportReq (SP) or
        IsAFAbortReq (SP) or
        IsAFAbortAndHeuristicReportReq (SP)
      ) and
      is_map_rollbackRI (SP)
    ) endfunc
function IsRollbackRsp (SP: ServicePrim) : bool is
    IsCRollbackRsp (SP) or
    ( ( IsAFHeuristicReportReq (SP) or
        IsAFAbortReq (SP) or
        IsAFAbortAndHeuristicReportReq (SP)
      ) and
      is_map_rollbackRI (SP)
    ) endfunc

function IsReq (SP: ServicePrim) : bool is
    IsTPDataReq (SP) or
    IsTPBeginDialogueReq (SP) or
    IsTPEndDialogueReq (SP) or
    IsTPUErrorReq (SP) or
    IsTPUAbortReq (SP) or
    IsTPGrantControlReq (SP) or
    IsTPRequestControlReq (SP) or
    IsTPHandshakeReq (SP) or
    IsTPHandshakeAndGrantControlReq (SP) or
    IsTPBeginTransactionReq (SP) or
    IsTPDeferredEndDialogueReq (SP) or
    IsTPDeferredGrantControlReq (SP) or
    IsTPCommitReq (SP) or
    IsTPDoneReq (SP) or
    IsTPPrepareReq (SP) or
    IsTPRollbackReq (SP) or
    IsAFBeginDialogueReq (SP) or
```

```
        IsAFBidReq (SP) or
        IsAFEndDialogueReq (SP) or
        IsAFUErrorReq (SP) or
        IsAFAbortReq (SP) or
        IsAFGrantControlReq (SP) or
        IsAFRequestControlReq (SP) or
        IsAFHandshakeReq (SP) or
        IsAFHandshakeAndGrantControlReq (SP) or
        IsAFDeferReq (SP) or
        IsAFPrepareReq (SP) or
        IsAFHeuristicReportReq (SP) or
        IsAFAbortAndHeuristicReportReq (SP) or
        IsAFTokenPleaseReq (SP) or
        IsAFTokenGiveReq (SP) or
        IsAFRecoverReq (SP) or
        IsCAFPleaseReq (SP) or
        IsCAFDetachReq (SP) or
        IsSAFDetachAssociationReq (SP) or
        IsCCommit_CBeginReq (SP) or
        IsCBeginReq (SP) or
        IsCPrepareReq (SP) or
        IsCReadyReq (SP) or
        IsCCommitReq (SP) or
        IsCRollbackReq (SP) or
        IsCRecoverReq (SP) or
        IsAAssocReq (SP) or
        IsAReleaseReq (SP) or
        IsAAbortReq (SP) or
        IsUaseReq (SP) or
        IsPDataReq (SP) or
        IsPTokenPleaseReq (SP) or
        IsPTokenGiveReq (SP)
    endfunc

    function IsRsp (SP: ServicePrim) : bool is
        IsTPBeginDialogueRsp (SP) or
        IsTPEndDialogueRsp (SP) or
        IsTPHandshakeRsp (SP) or
        IsTPHandshakeAndGrantControlRsp (SP) or
        IsAFBeginDialogueRsp (SP) or
        IsAFBidRsp (SP) or
        IsAFEndDialogueRsp (SP) or
        IsAFUErrorRsp (SP) or
        IsAFHandshakeRsp (SP) or
        IsAFHandshakeAndGrantControlRsp (SP) or
        IsCBeginRsp (SP) or
        IsCCommitRsp (SP) or
        IsCRollbackRsp (SP) or
        IsCRecoverRsp (SP) or
        IsAAssocRsp (SP) or
        IsAReleaseRsp (SP)
    endfunc
    function IsInd (SP: ServicePrim) : bool is
        IsTPDataInd (SP) or
```

```
      IsTPBeginDialogueInd (SP) or
      IsTPEndDialogueInd (SP) or
      IsTPUErrorInd (SP) or
      IsTPUAbortInd (SP) or
      IsTPPAbortInd (SP) or
      IsTPGrantControlInd (SP) or
      IsTPRequestControlInd (SP) or
      IsTPHandshakeInd (SP) or
      IsTPHandshakeAndGrantControlInd (SP) or
      IsTPHeuristicReportInd (SP) or
      IsTPBeginTransactionInd (SP) or
      IsTPDeferredEndDialogueInd (SP) or
      IsTPDeferredGrantControlInd (SP) or
      IsTPCommitInd (SP) or
      IsTPCommitCompleteInd (SP) or
      IsTPPrepareInd (SP) or
      IsTPReadyInd (SP) or
      IsTPRollbackInd (SP) or
      IsTPRollbackCompleteInd (SP) or
      IsAFBeginDialogueInd (SP) or
      IsAFBidInd (SP) or
      IsAFEndDialogueInd (SP) or
      IsAFUErrorInd (SP) or
      IsAFAbortInd (SP) or
      IsAFGrantControlInd (SP) or
      IsAFRequestControlInd (SP) or
      IsAFHandshakeInd (SP) or
      IsAFHandshakeAndGrantControlInd (SP) or
      IsAFDeferInd (SP) or
      IsAFPrepareInd (SP) or
      IsAFHeuristicReportInd (SP) or
      IsAFAbortAndHeuristicReportInd (SP) or
      IsAFTokenPleaseInd (SP) or
      IsAFTokenGiveInd (SP) or
      IsAFRecoverInd (SP) or
      IsCAFGiveInd (SP) or
      IsCAFRecoverInd (SP) or
      IsSAFAssociationLostInd (SP) or
      IsCCommit_CBeginInd (SP) or
      IsCBeginInd (SP) or
      IsCPrepareInd (SP) or
      IsCReadyInd (SP) or
      IsCCommitInd (SP) or
      IsCRollbackInd (SP) or
      IsCRecoverInd (SP) or
      IsAAssocInd (SP) or
      IsAReleaseInd (SP) or
      IsAAbortInd (SP) or
      IsAPAbortInd (SP) or
      IsUaseInd (SP) or
      IsPDataInd (SP) or
      IsPTokenPleaseInd (SP) or
      IsPTokenGiveInd (SP)
  endfunc
```

```
function IsCnf (SP: ServicePrim) : bool is
    IsTPBeginDialogueCnf (SP) or
    IsTPEndDialogueCnf (SP) or
    IsTPHandshakeCnf (SP) or
    IsTPHandshakeAndGrantControlCnf (SP) or
    IsAFBeginDialogueCnf (SP) or
    IsAFBidCnf (SP) or
    IsAFEndDialogueCnf (SP) or
    IsAFUErrorCnf (SP) or
    IsAFHandshakeCnf (SP) or
    IsAFHandshakeAndGrantControlCnf (SP) or
    IsCBeginCnf (SP) or
    IsCCommitCnf (SP) or
    IsCRollbackCnf (SP) or
    IsCRecoverCnf (SP) or
    IsAAssocCnf (SP) or
    IsAReleaseCnf (SP)
endfunc
function IsReqRsp (SP: ServicePrim) : bool is
    IsReq (SP) or
    IsRsp (SP) or
    IsPWithEmbeddedAPDUReqRsp (SP) or
    IsOtherPresentationReqRsp (SP)
endfunc
function IsIndCnf (SP: ServicePrim) : bool is
    IsInd (SP) or
    IsCnf (SP) or
    IsPWithEmbeddedAPDUIndCnf (SP) or
    IsOtherPresentationIndCnf (SP)
endfunc

function is_user (SP: ServicePrim) : bool is (get_type (SP) match user endfunc
function is_provider (SP: ServicePrim) : bool is (get_type (SP) match provider endfunc
function is_absent (SP: ServicePrim) : bool is SP match absentSP endfunc

function get_type (SP: ServicePrim) : tp_abort_type is
      if IsTPUAbortReq (SP) or IsTPUAbortInd (SP) then user
      elsif IsTPPAbortInd (SP) then provider
      endif
endfunc

ofsort address
function get_addr (SP: ServicePrim) : address is
      if IsTPBeginDialogueReq (SP) then
          address(Opt(get_rapt (SP)), get_rapiO (SP), get_raeqO (SP), get_raeiO (SP))
      elsif IsTPBeginDialogueInd (SP) then
          address(get_iaptO (SP), get_iapiO (SP), get_iaeqO (SP), get_iaeiO (SP))
      endif
endfunc

function CBeginReq_CRecoverRspUnknown (SPK: ServicePrimKey,
      aaid: atomic_action_identifier,
      brid: branch_identifier) : ServicePrim is
      case SPK in
```

```
            CBegin_req -> CBeginReq (aaid, brid)
        |   CRecover_rsp -> CRecoverRsp(unknown, aaid, brid)
        endcase
  endfunc

  function CRecover_AFRecover_Req (rsri: recovery_state,
        rchO: recovery_context_handle_Opt,
        aaid: atomic_action_identifier,
        brid: branch_identifier) : ServicePrim is
        case rchO in
            rch_absent -> CRecoverReq(rsri, aaid, brid)
        |   Opt(RCH:=rch: recovery_context_handle)
                -> AFRecoverReq(make_map(rsri), rch, aaid, brid)
        endcase
  endfunc

  function key (SP: ServicePrim) : ServicePrimKey is
        case SP in
            TP (TP_SP: TP_ServicePrim) -> key (TP_SP)
        |   AF (AF_SP: AF_ServicePrim) -> key (AF_SP)
        |   CAF (CAF_SP: CAF_ServicePrim) -> key (CAF_SP)
        |   SAF (SAF_SP: SAF_ServicePrim) -> key (SAF_SP)
        |   CCR (CCR_SP: CCR_ServicePrim) -> key (CCR_SP)
        |   ACSE (ACSE_SP: ACSE_ServicePrim) -> key (ACSE_SP)
        |   P (P_SP: P_ServicePrim) -> key (P_SP)
        |   absentSP -> succ(last_Psp)
  endfunc
endmod
=================================================================== *)

(*
 *      H.5.5   MACF Data Structures
 *)
type dialogue_channel_identifiers is NaturalNumber
sorts dc_id, set_of_dids
opns
(* dc_id *)
  dial (*! constructor *),
  chan (*! constructor *) : nat -> dc_id
  sup_dial, first_sub_dial, first_chan,
  did_absent (*! constructor *) : -> dc_id
  inc : dc_id -> dc_id
  _eq_, _ne_, _lt_, _le_ : dc_id, dc_id -> bool
  is_dial, is_chan, is_absent, is_superior_dial, is_subordinate_dial :
    dc_id -> bool

(* set_of_dids *)
  empty_dids (*! constructor *) : -> set_of_dids
  _+_ (*! constructor *), _-_ : set_of_dids, dc_id -> set_of_dids
  IsEmpty : set_of_dids -> bool
  _+_, _-_ : set_of_dids, set_of_dids -> set_of_dids
  _isin_ : dc_id, set_of_dids -> bool
eqns
    forall did, did1 : dc_id, did_set, did_set1 : set_of_dids, n, m : nat
  ofsort bool
    dial(n) eq dial(m) = n eq m;
    dial(n) eq chan(m) = false;
    dial(n) eq did_absent = false;
    chan(n) eq dial(m) = false;
    chan(n) eq chan(m) = n eq m;
    chan(n) eq did_absent = false;
    did_absent eq dial(m) = false;
    did_absent eq chan(m) = false;
    did_absent eq did_absent = true;
    did ne did1 = not(did eq did1);
    dial(n) lt dial(m) = n lt m;
    dial(n) lt chan(m) = true;
    dial(n) lt did_absent = true;
    chan(n) lt dial(m) = false;
    chan(n) lt chan(m) = n lt m;
    chan(n) lt did_absent = true;
    did_absent lt did = false;
    did le did1 = not(did1 lt did1);
    did IsIn empty_dids = false;
    did IsIn (did_set + did1) = (did eq did1) or (did IsIn did_set);
    IsEmpty(empty_dids) = true;
    IsEmpty(did_set + did) = false;
    is_dial(did) = did lt first_chan;
    is_chan(did) = not(is_dial(did) or is_absent(did));
    is_absent(did) = did eq did_absent;
    is_superior_dial(did) = did eq sup_dial;
    is_subordinate_dial(did) = is_dial(did) and not(is_superior_dial(did))
  ofsort dc_id
    sup_dial = dial(0);
    first_sub_dial = dial(1);
    first_chan = chan(0);
    inc(dial(n)) = dial(succ(n));
    inc(chan(n)) = chan(succ(n));
    inc(did_absent) = did_absent;
  ofsort set_of_dids
    empty_dids - did = empty_dids;
    (did_set + did) - did = did_set;
    did ne did1 => (did_set + did) - did1 = (did_set - did1) + did;
    did_set + empty_dids = did_set;
    did IsIn did_set => did_set + (did_set1 + did) = did_set + did_set1;
    not(did IsIn did_set) =>
      did_set + (did_set1 + did) = (did_set + did_set1) + did;
    did_set - empty_dids = did_set;
(*  did_set - (did_set1 + did) = (did_set - did_set1) - did_set; *)
    did_set - (did_set1 + did) = (did_set - did) - did_set1;
endtype
(* ------------------------------------------------------------------------
module dialogue_channel_identifiers is NaturalNumber
    type dc_id is
        dial (N: nat)
    |   chan (N: nat)
    |   did_absent
    endtype
    function sup_dial : dc_id is dial(0) endfunc
    function first_sub_dial : dc_id is dial(1) endfunc
    function first_chan : dc_id is chan(0) endfunc
    function inc (d: dc_id) : dc_id is
        case d is
            did_absent -> did_absent
        |   dial (N:=n: nat) -> dial (succ (n))
        |   chan (N:=n: nat) -> chan (succ (n))
        endcase
    endfunc

  is_dial, is_chan, is_absent, is_superior_dial, is_subordinate_dial :
    dc_id -> bool

/* syntactical equality will be automatically generated */
    function _ne_ (d1: dc_id, d2: dc_id) : bool is not (d1 eq d2) endfunc
    function _lt_ (d1: dc_id, d2: dc_id) : bool is
        case d1 in
          did_absent -> false
        | dial (N:=n: nat) ->   case d2 in
                                    dial (N:=m: nat) -> n lt m
                                |   any dc_id  -> true endcase
        | chan (N:=n: nat) ->   case d2 in
                                    did_absent -> true
                                |   dial (...) -> false
                                |   chan (N:=m: nat) -> n lt m endcase
        endcase
    endfunc

    function _le_ (d1: dc_id, d2: dc_id) : bool is (d1 lt d2) or (d1 eq d2) endfunc
    function is_dial (d: dc_id) : bool is d match dial (...) endfunc
    function is_chan (d: dc_id) : bool is d match chan (...) endfunc
    function is_absent (d: dc_id) : bool is d match did_absent endfunc
    function is_superior_dial (d: dc_id) : bool is d eq sup_dial endfunc
    function is_subordinate_dial (d: dc_id) : bool is
        is_dial (d) and not ((select N in d) eq 0) endfunc

/* set_od_dids */
    type set_of_dids is
        empty_dids
    |   _+_ (S: set_of_dids, D: dc_id)
    endtype

    function _-_ (S1: set_of_dids, D1: dc_id) : set_of_dids is
        case S1 in
            empty_dids -> empty_dids
        |   (S:=S2: set_of_dids) + (D:=D2: dc_id)
                -> if D1 eq D2 then D2 else (S2 - D1) + D2 endif
        endcase
    endfunc

    function IsEmpty (S1: set_of_dids) : bool is S1 match empty_dids endfunc

    function _isin_ (D1: dc_id, S1:set_of_dids) : bool is
```

```
        case S1 in
            empty_dids -> false
        |   (S:=S2: set_of_dids) + (D:=D2: dc_id) -> (D1 eq D2) or (D1 isin S2)
        endcase
    endfunc

    function _+_ (S1: set_of_dids, S2: set_of_dids) : set_of_dids is
        case S2 in
            empty_dids -> S1
        |   (S:=DS: set_of_dids) + (D:=D2: dc_id) ->
                if D2 IsIn S1 then S1 + DS else (S1 + DS) + D2 endif
        endcase
    endfunc

    function _-_  (S1: set_of_dids, S2: set_of_dids) : set_of_dids is
        case S2 in
            empty_dids -> S1
        |   (S:=DS: set_of_dids) + (D:=D2: dc_id) -> (S1 - D2) - DS
        endcase
    endfunc

endmod
======================================================================== *)


type log_records is association_info
sorts log_record, bound_data_state, branch_list, branch_record
opns
(* log_record *)
  log_record (*! constructor *) :
    recovery_state, atomic_action_identifier, branch_list -> log_record
  get_state : log_record -> recovery_state
  get_aaid : log_record -> atomic_action_identifier
  get_branch_record : log_record -> branch_list
  get_branch : branch_identifier, log_record -> branch_record
  get_branch : branch_identifier, branch_list -> branch_record
  branch_list, branch_records : set_of_assocs -> branch_list
  branch_record : assoc_entry -> branch_record
  branch_record : dc_id, set_of_assocs -> branch_record
  recreate_assocs : log_record -> set_of_assocs
  recreate_assocs : dc_id, branch_list -> set_of_assocs

(* bound_data_state *)
  initial (*! constructor *),
  final (*! constructor *),
  ready (*! constructor *),
  mixed (*! constructor *) : -> bound_data_state
  is_ready, is_initial, is_final : bound_data_state -> bool
  data_nat : bound_data_state -> nat

(* branch_list *)
  _._ (*! constructor *) : branch_record, branch_list -> branch_list
  nil_branch (*! constructor *) : -> branch_list
  IsEmpty : branch_list -> bool
  sup_record : branch_list -> branch_record
(* branch_record *)
  branch (*! constructor *) :
    branch_identifier, AE_Title, recovery_context_handle_Opt -> branch_record
  sup_branch (*! constructor *) : branch_identifier, recovery_context_handle_Opt -> branch_record
  get_brid : branch_record -> branch_identifier
  get_aet : branch_record -> AE_Title
  get_rchO : branch_record -> recovery_context_handle_Opt
  has_branch : branch_identifier, branch_list -> bool
eqns
    forall
      log : log_record,
      rsri : recovery_state,
      aaid, aaid1 : atomic_action_identifier,
      bds : bound_data_state,
      br : branch_record,
      bl : branch_list,
      a_set : set_of_assocs,
      ent : assoc_entry,
      did : dc_id,
      brid : branch_identifier,
      aet : AE_title,
      rchO : recovery_context_handle_Opt
  ofsort bool
    is_initial(bds) = data_nat(bds) eq 0;
    is_ready(bds) = data_nat(bds) eq 1;
    is_final(bds) = data_nat(bds) eq 2;
    IsEmpty(nil_branch) = true;
    IsEmpty(br . bl) = false;
  ofsort recovery_state get_state(log_record(rsri, aaid, bl)) = rsri;
  ofsort atomic_action_identifier get_aaid(log_record(rsri, aaid, bl)) = aaid;
  ofsort set_of_assocs
    is_ready(get_state(log)) =>
      recreate_assocs(log) = recreate_assocs(sup_dial, get_branch_record(log));
    is_commit(get_state(log)) =>
      recreate_assocs(log) =
        recreate_assocs(first_sub_dial, get_branch_record(log));
    recreate_assocs(did, nil_branch) = empty_assocs;
    recreate_assocs(did, br . bl) =
      recreate_assocs(inc(did), bl) +
      assoc( did, aet_absent, rch_absent, Opt(get_brid(br)), nilSP, nilSP,
        absentSP, purge_absent, false, true)
  ofsort branch_record
    sup_record(branch(brid, aet, rchO) . bl) = sup_record(bl);
    sup_record(sup_branch(brid, rchO) . bl) = sup_branch(brid, rchO);
    branch_record(did, a_set) = branch_record(get_assoc(did, a_set));
    is_superior(ent) =>
      branch_record(ent) = sup_branch(get_brid(ent), get_rchO(ent));
    is_subordinate(ent) =>
      branch_record(ent) = branch(get_brid(ent), get_aet(ent), get_rchO(ent));
    get_branch(brid, log) = get_branch(brid, get_branch_record(log));
    get_brid(br) eq brid => get_branch(brid, br . bl) = br;
    not(get_brid(br) eq brid) =>
      get_branch(brid, br . bl) = get_branch(brid, bl);
  ofsort branch_list
    get_branch_record(log_record(rsri, aaid, bl)) = bl;
    branch_records(empty_assocs) = nil_branch;
    branch_records(a_set + ent) = branch_record(ent) . branch_records(a_set);
    branch_list(a_set) = branch_records(branches(a_set));
  ofsort branch_identifier
    get_brid(branch(brid, aet, rchO)) = brid;
    get_brid(sup_branch(brid, rchO)) = brid;
  ofsort AE_title
    get_aet(branch(brid, aet, rchO)) = aet;
    get_aet(sup_branch(brid, rchO)) = superiors_name(brid);
  ofsort recovery_context_handle_Opt
    get_rchO(branch(brid, aet, rchO)) = rchO;
    get_rchO(sup_branch(brid, rchO)) = rchO;
  ofsort nat
    data_nat(initial) = 0;
    data_nat(ready) = 1;
    data_nat(final) = 2;
    data_nat(mixed) = 3;
  ofsort bool
    has_branch(brid, br . bl) = (brid eq get_brid(br)) or has_branch(brid, bl);
    has_branch(brid, nil_branch) = false;
endtype
(* ------------------------------------------------------------------------
module log_records is association_info

    type log_record is
        log_record (RSRI: recovery_state,
                AAID: atomic_action_identifier,
                BRID: branch_list)
    endtype

    function get_state (LR: log_record) : recovery_state is select RSRI in LR endfunc
    function get_aaid (LR: log_record) : recovery_state is select AAID in LR endfunc
    function get_branch_record (LR: log_record) : recovery_state is select BRID in LR endfunc


/* bound_data_state */
    type bound_data_state is
        initial
    |   final
    |   ready
    |   mixed
    endfunc

    function is_ready (BDS: bound_data_state) : bool is BDS match ready endfunc
    function is_initial (BDS: bound_data_state) : bool is BDS match initial endfunc
    function is_final (BDS: bound_data_state) : bool is BDS match final endfunc
/* data_nat function  translation is not needed */

/* branch_record */
    type branch_record is
        branch (BRID: branch_identifier,
                AET: AE_Title,
                RCHO: recovery_context_handle_Opt)
    |   sup_branch (BRID: branch_identifier,
                RCHO: recovery_context_handle_Opt)
    endtype

    function get_brid (BR: branch_record) : branch_identifier is select BRID in BR endfunc
    function get_aet (BR: branch_record) : AE_Title is select AET in CR endfunc
    function get_rchO (BR: branch_record) : recovery_context_handle_Opt is select RCHO in BR endfunc

/* branch_list */
    type branch_list i
        nil_branch
    |   _._ (BR: branch_record, BL: branch_list)
    endtype

    function IsEmpty (L: branch_list) : bool is L match nil_branch endfunc

    function sup_record (L: branch_list) : branch_record is
        case (select BR in L) in
            branch (...) -> sup_record (select BL in L)
        |   sup_branch (...) -> select BR in L
        endcase
    endfunc

    function get_branch (BI: branch_identifier, L: branch_list) : branch_record is
        if (get_brid (select BR in L) eq BI) then (select BR in L)
        else get_branch (BI, (select BL in L)) endif
    endfunc

    function get_branch (BI: branch_identifier, LR: log_record) : branch_record is
        get_branch (BI, get_branch_record (LR)) endfunc

    function branch_list (a_set: set_of_assocs) : branch_list is
        branch_records(branches(a_set)) endfunc

    function branch_records (a_set: set_of_assocs) : branch_list is
        case a_set in
            empty_assocs -> nil_branch
        |   (S: set_of_assocs) + (E: assoc_entry) -> branch_record(E) . branch_records(S)
        endcase
    endfunc

    function branch_record (E: assoc_entry) : branch_record is
        if is_superior (E) then
                sup_branch (get_brid (E), get_rchO (E))
        elsif is_subordinate (E) then
                branch (get_brid(E), get_aet (E), get_rchO (E)) endif
    endfunc

    function branch_record (D: dc_id, S: set_of_assocs) : branch_record is
        branch_record (get_assoc (D, S)) endfunc

    function recreate_assocs (L: log_record) : set_of_assocs is
        if is_ready (get_state (L)) then
```

```
                recreate_assocs(sup_dial, get_branch_record(L))
        elsif is_commit (get_state (L)) then
                recreate_assocs (first_sub_dial, get_branch_record (L))
        endif
    endfunc

    function recreate_assocs (D: dc_id, BL: branch_list) : set_of_assocs is
        case BL in
            nil_branch -> empty_assocs
        |   (BR:=br: branch_record) . (BL:=L:  branch_list) ->
                recreate_assocs (inc(D), L) +
                assoc( D, aet_absent, rch_absent, Opt (get_brid (br)),
                nilSP, nilSP, absentSP, purge_absent, false, true)
        endcase
    endfunc

    function has_branch (BI: branch_identifier, L: branch_list) : bool is
        case L in
            nil_branch -> false
        |   (BR:=br: branch_record, BL:=bl: branch_list) ->
                (BI eq get_brid (br)) or has_branch(BI, bl)
        endcase
    endfunc
endmod
========================================================================= *)


type association_info is
  dialogue_channel_identifiers, local_values, ServicePrim_Lists
sorts assoc_entry, purge_sort, set_of_assocs
opns
(* assoc_entry *)
  assoc (*! constructor *) :
        dc_id,
        AE_title_Opt, (* comments shifted *)
        recovery_context_handle_Opt,
        branch_identifier_Opt, (* comments shifted *)
        ServicePrimList,
        (*  sent list  *) ServicePrimList,
        (*  pending list  *) ServicePrim,
        (*  initializing ServicePrim  *) purge_sort,
        (*  TPUErrorReq purging count: dialogue only  *) bool,
        bool (* comment shifted *) -> assoc_entry

(* set_of_assocs *)
  empty_assocs (*! constructor *) : -> set_of_assocs
  _+_ (*! constructor *) : set_of_assocs, assoc_entry -> set_of_assocs
  add : assoc_entry, set_of_assocs -> set_of_assocs
  IsEmpty : set_of_assocs -> bool
  match : dc_id, assoc_entry -> bool
  remove : dc_id, set_of_assocs -> set_of_assocs
  remove : set_of_assocs, set_of_assocs -> set_of_assocs
  get_assoc : dc_id, set_of_assocs -> assoc_entry
  get_aet : dc_id, set_of_assocs -> AE_title
```

```
get_SentRcvd : dc_id, set_of_assocs -> ServicePrimList
get_SentRcvd : LookUpKey, dc_id, set_of_assocs -> ServicePrim
get_Pending : dc_id, set_of_assocs -> ServicePrimList
get_Pending : LookUpKey, dc_id, set_of_assocs -> ServicePrim
get_brid : dc_id, set_of_assocs -> branch_identifier
get_rchO : dc_id, set_of_assocs -> recovery_context_handle_Opt
get_sfu : dc_id, set_of_assocs -> (*selected*) functional_units
get_bdc : dc_id, set_of_assocs -> tp_begin_dialogue_confirmation
get_cu : dc_id, set_of_assocs -> channel_utilization
is_did_in,
is_dial_in,
is_chan_in,
is_superior,
is_subordinate,
is_coord_commitment,
is_coord_none,
is_detached,
is_purging,
is_trans_init_purging,
transfered_to_another_tppm,
is_free_channel,
retry,
is_chaining :
  dc_id, set_of_assocs -> bool
is_in_SentRcvd, is_in_Pending, is_last :
  LookUpKey, dc_id, set_of_assocs -> bool
get_attached_branch, get_branch_entry :
  branch_identifier, set_of_assocs -> dc_id
get_sup : set_of_assocs -> dc_id
get_did : assoc_entry -> dc_id
get_aet : assoc_entry -> AE_title
get_SentRcvd : assoc_entry -> ServicePrimList
get_Pending : assoc_entry -> ServicePrimList
get_brid : assoc_entry -> branch_identifier
get_rchO : assoc_entry -> recovery_context_handle_Opt
get_sfu : assoc_entry -> (*selected*) functional_units
get_bdc : assoc_entry -> tp_begin_dialogue_confirmation
get_cu : assoc_entry -> channel_utilization
is_superior, is_subordinate : branch_identifier -> bool
is_dialogue,
is_channel,
is_superior,
is_subordinate,
attached,
is_purging,
flag,
is_trans_init_purging,
is_free_channel,
has_brid,
is_chaining : assoc_entry -> bool
is_in_SentRcvd, is_in_Pending : LookUpKey, assoc_entry -> bool
has_attached_branch : branch_identifier, set_of_assocs -> bool
remove_branch_entries : branch_identifier, set_of_assocs -> set_of_assocs
prune_branch_entries, set_brid :
  branch_identifier, dc_id, set_of_assocs -> set_of_assocs
remove_SentRcvd, remove_Pending, change_pending_to_sent :
  LookUpKey, dc_id, set_of_assocs -> set_of_assocs
add_SentRcvd, add_Pending :
  ServicePrim, dc_id, set_of_assocs -> set_of_assocs
add_SentRcvd : ServicePrim, set_of_assocs, set_of_assocs -> set_of_assocs
detach,
coord_commitment,
coord_none,
retry_needed,
retry_sent,
end_purge,
trans_init,
inc_purge,
dec_purge :
  dc_id, set_of_assocs -> set_of_assocs
transfer : bool, dc_id, set_of_assocs -> set_of_assocs
re_attach : ServicePrim, dc_id, set_of_assocs -> set_of_assocs
set_brid : branch_identifier, assoc_entry -> assoc_entry
remove_SentRcvd, remove_Pending, change_pending_to_sent :
  LookUpKey, assoc_entry -> assoc_entry
re_attach, add_SentRcvd, add_Pending :
  ServicePrim, assoc_entry -> assoc_entry
set_flag : bool, assoc_entry -> assoc_entry
detach, remove_commitSPs, end_purge, trans_init, inc_purge, dec_purge :
  assoc_entry -> assoc_entry
dialogue_entry :
  dc_id, address, recovery_context_handle_Opt, ServicePrim -> assoc_entry
channel_entry : dc_id, address, ServicePrim -> assoc_entry
recovery_entry : dc_id, branch_identifier, ServicePrim -> assoc_entry
has_sup,
is_root,
is_intermediate,
is_leaf,
is_root_or_intermediate,
is_intermediate_or_leaf,
any_detached,
any_chaining,
rollback_rep_compl : set_of_assocs -> bool
init_Assocs,
dialogues,
channels,
branches,
subordinates,
attached,
remove_commitSPs,
IsTwoWayChannel,
chaining,
chaining_subordinates,
polarized_selected,
shared_selected,
unchained_selected : set_of_assocs -> set_of_assocs
HasSentRcvd, HasPending, NotSentRcvd, NoPending : LookUpKey, set_of_assocs -> set_of_assocs
get_dids : set_of_assocs -> set_of_dids
next_chid : set_of_assocs -> dc_id

(* purge_sort *)
purging (*! constructor *) : nat, (* shared  *) nat (* trans_init purging count  *) -> purge_sort
purging (*! constructor *) : bool -> purge_sort (* polarized  *)
purge_absent (*! constructor *) : -> purge_sort
(* used on channels where there is no purging  *)
init_purge : functional_units -> purge_sort
inc, dec, end : purge_sort -> purge_sort
is_purging : purge_sort -> bool
is_trans_init : purge_sort -> bool
trans_init : purge_sort -> purge_sort
eqns
  forall
    ent : assoc_entry,
    a_set, set : set_of_assocs,
    did : dc_id,
    aetO : AE_title_Opt,
    aet : AE_title,
    addr : address,
    sent, pend : ServicePrimList,
    brid : branch_identifier,
    bridO : branch_identifier_Opt,
    rchO : recovery_context_handle_Opt,
    purge : purge_sort,
    sp, init_sp : ServicePrim,
    x : LookUpKey,
    flag, new, attach : bool
ofsort bool
  IsEmpty(empty_assocs) = true;
  IsEmpty(a_set + ent) = false;
  match(did, ent) = did eq get_did(ent);
ofsort set_of_assocs
  remove(did, empty_assocs) = empty_assocs;
  match(did, ent) => remove(did, a_set + ent) = a_set;
  not(match(did, ent)) => remove(did, a_set + ent) = remove(did, a_set) + ent;
  remove_branch_entries(brid, empty_assocs) = empty_assocs;
  has_brid(ent) and (get_brid(ent) eq brid) =>
    remove_branch_entries(brid, a_set + ent) = remove_branch_entries(brid, a_set);
  not(has_brid(ent) and (get_brid(ent) eq brid)) =>
    remove_branch_entries(brid, a_set + ent) = remove_branch_entries(brid, a_set) + ent;
ofsort assoc_entry
  match(did, ent) => get_assoc(did, a_set + ent) = ent;
  not(match(did, ent)) => get_assoc(did, a_set + ent) = get_assoc(did, a_set);
ofsort dc_id
  get_did( assoc(did, aetO, rchO, bridO, sent, pend, init_sp, purge, flag, attach)) = did;
ofsort AE_title
  get_aet(assoc(did, Opt(aet), rchO, bridO, sent, pend, init_sp, purge, flag, attach)) = aet;
  get_aet(did, a_set) = get_aet(get_assoc(did, a_set));
ofsort branch_identifier
  get_brid(assoc( did, aetO, rchO, Opt(brid), sent, pend, init_sp, purge, flag, attach)) = brid;
  get_brid(did, a_set) = get_brid(get_assoc(did, a_set));
ofsort ServicePrimList
  get_SentRcvd(assoc(did, aetO, rchO, bridO, sent, pend, init_sp, purge, flag, attach)) = sent;
  get_SentRcvd(did, a_set) = get_SentRcvd(get_assoc(did, a_set));
  get_Pending(assoc(did, aet0, rch0, brid0, sent, pend, init_sp, purge, flag, attach)) = pend;
  get_Pending(did, a_set) = get_Pending(get_assoc(did, a_set));
ofsort recovery_context_handle_Opt
  get_rch0(assoc(did, aet0, rch0, brid0, sent, pend, init_sp, purge, flag, attach)) = rch0;
  get_rch0(did, a_set) = get_rch0(get_assoc(did, a_set));
ofsort functional_units
  get_sfu(assoc(did, aet0, rch0, brid0, sent, pend, init_sp, purge, flag, attach)) = get_sfu(init_sp);
  get_sfu(did, a_set) = get_sfu(get_assoc(did, a_set));
ofsort tp_begin_dialogue_confirmation
  get_bdc(assoc(did, aet0, rch0, brid0, sent, pend, init_sp, purge, flag, attach)) = get_bdc(init_sp);
  get_bdc(did, a_set) = get_bdc(get_assoc(did, a_set));
ofsort channel_utilization
  get_cu(assoc(did, aet0, rch0, brid0, sent, pend, init_sp, purge, flag, attach)) =
    get_cu(init_sp);
  get_cu(did, a_set) = get_cu(get_assoc(did, a_set));
ofsort ServicePrim
  get_SentRcvd(x, did, a_set) = get(x, get_SentRcvd(did, a_set));
  get_Pending(x, did, a_set) = get(x, get_Pending(did, a_set));
ofsort bool
  is_did_in(did, empty_assocs) = false;
  is_did_in(did, a_set + ent) = match(did, ent) or is_did_in(did, a_set);
  is_dialogue(ent) = is_dial(get_did(ent)) and attached(ent);
  is_dial_in(did, a_set) = is_did_in(did, a_set) and is_dialogue(get_assoc(did, a_set));
  is_channel(ent) = is_chan(get_did(ent)) and attached(ent);
  is_chan_in(did, a_set) = is_did_in(did, a_set) and is_channel(get_assoc(did, a_set));
  flag(assoc(did, aet0, rch0, brid0, sent, pend, init_sp, purge, flag, attach)) = flag;
  attached(assoc(did, aet0, rch0, brid0, sent, pend, init_sp, purge, flag, attach)) = attach;
  is_coord_commitment(did, a_set) = flag(get_assoc(did, a_set));
  is_coord_none(did, a_set) = not(is_coord_commitment(did, a_set));
  is_detached(did, a_set) = not(is_did_in(did, a_set)) or not(attached(get_assoc(did, a_set)));
  transfered_to_another_tppm(did, a_set) =
    is_did_in(did, a_set) and
    not(attached(get_assoc(did, a_set))) and
    flag(get_assoc(did, a_set));
  retry(did, a_set) =
    is_did_in(did, a_set) and
    not(attached(get_assoc(did, a_set))) and
    flag(get_assoc(did, a_set));
  is_free_channel(ent) =
    attached(ent) and not(is_in_SentRcvd(CAFPlease_requ, ent));
  is_free_channel(did, a_set) =
    is_did_in(did, a_set) and is_free_channel(get_assoc(did, a_set));
  is_purging(assoc(did, aet0, rch0, brid0, sent, pend, init_sp, purge, flag, attach)) =
    is_purging(purge);
  is_purging(did, a_set) = is_purging(get_assoc(did, a_set));
  is_trans_init_purging(assoc(did, aet0, rch0, brid0, sent, pend, init_sp, purge, flag, attach)) =
    is_trans_init(purge);
  is_trans_init_purging(did, a_set) = is_trans_init_purging(get_assoc(did, a_set));
  is_last(x, did, a_set) = last(get_SentRcvd(did, a_set)) eq x;
  has_brid(assoc(did, aet0, rch0, Opt(brid), sent, pend, init_sp, purge, flag, attach)) = true;
  has_brid(assoc(did, aet0, rch0, brid_absent, sent, pend, init_sp, purge, flag, attach)) = false;
  is_chaining(did, a_set) = is_did_in(did, a_set) and is_chaining(get_assoc(did, a_set));
  is_chaining(ent) = attached(ent) and chained_selected(get_sfu(ent))
```

```
      and not(is_in_SentRcvd(DeferEndDialogue, ent));
    is_in_SentRcvd(x, did, a_set) = x IsIn get_SentRcvd(did, a_set);
    is_in_SentRcvd(x, ent) = x IsIn get_SentRcvd(ent);
    is_in_Pending(x, did, a_set) = x IsIn get_Pending(did, a_set);
    is_in_Pending(x, ent) = x IsIn get_Pending(ent);
    is_superior(brid) = not(is_subordinate(brid));
    is_subordinate(brid) = superiors_name(brid) eq my_aet;
    is_superior(ent) = has_brid(ent) and is_superior(get_brid(ent));
    is_subordinate(ent) = has_brid(ent) and is_subordinate(get_brid(ent));
    is_superior(did, a_set) = is_superior(get_assoc(did, a_set));
    is_subordinate(did, a_set) = is_subordinate(get_assoc(did, a_set));
    has_sup(a_set) = not(is_absent(get_sup(a_set)));
    is_root(a_set) = not(has_sup(a_set));
    is_intermediate(a_set) = is_root_or_intermediate(a_set) and is_intermediate_or_leaf(a_set);
    is_leaf(a_set) = IsEmpty(subordinates(a_set));
    is_root_or_intermediate(a_set) = not(is_leaf(a_set));
    is_intermediate_or_leaf(a_set) = has_sup(a_set);
    any_detached(a_set) = not(IsEmpty(remove(attached(a_set), a_set)));
    any_chaining(a_set) = not(IsEmpty(chaining(a_set)));
    rollback_rep_compl(a_set) = not(has_sup(a_set)) or
      is_in_SentRcvd(RollbackRsp, sup_dial, a_set) or
      is_in_SentRcvd(RollbackCnf, sup_dial, a_set);
    has_attached_branch(brid, a_set) = not(is_absent(get_attached_branch(brid, a_set)));
  ofsort dc_id
    get_branch_entry(brid, empty_assocs) = did_absent;
    has_brid(ent) and (get_brid(ent) eq brid) => get_branch_entry(brid, a_set + ent) = get_did(ent);
    not(has_brid(ent) and (get_brid(ent) eq brid)) =>
      get_branch_entry(brid, a_set + ent) = get_branch_entry(brid, a_set);
    get_attached_branch(brid, empty_assocs) = did_absent;
    attached(ent) and has_brid(ent) and (get_brid(ent) eq brid) =>
      get_attached_branch(brid, a_set + ent) = get_did(ent);
    not(attached(ent) and has_brid(ent) and (get_brid(ent) eq brid)) =>
      get_attached_branch(brid, a_set + ent) = get_attached_branch(brid, a_set);
    get_sup(empty_assocs) = did_absent;
    is_superior(ent) => get_sup(a_set + ent) = get_did(ent);
    not(is_superior(ent)) => get_sup(a_set + ent) = get_sup(a_set);
  ofsort set_of_assocs
    add(ent, a_set) = a_set + ent;
    set_brid(brid, did, a_set) = remove(did, a_set) + set_brid(brid, get_assoc(did, a_set));
    add_SentRcvd(sp, did, a_set) = remove(did, a_set) + add_SentRcvd(sp, get_assoc(did, a_set));
    add_SentRcvd(sp, empty_assocs, a_set) = a_set;
    add_SentRcvd(sp, set + ent, a_set) =
      add_SentRcvd(sp, set, add_SentRcvd(sp, get_did(ent), a_set));
    change_pending_to_sent(x, did, a_set) =
      remove(did, a_set) + change_pending_to_sent(x, get_assoc(did, a_set));
    add_Pending(sp, did, a_set) =
      remove(did, a_set) + add_Pending(sp, get_assoc(did, a_set));
    remove_SentRcvd(x, did, a_set) =
      remove(did, a_set) + remove_SentRcvd(x, get_assoc(did, a_set));
    remove_Pending(x, did, a_set) =
      remove(did, a_set) + remove_Pending(x, get_assoc(did, a_set));
    coord_commitment(did, a_set) =
      remove(did, a_set) + set_flag(true, get_assoc(did, a_set));
    coord_none(did, a_set) =
      remove(did, a_set) + set_flag(false, get_assoc(did, a_set));
    retry_needed(did, a_set) =
      remove(did, a_set) + set_flag(true, get_assoc(did, a_set));
    retry_sent(did, a_set) =
      remove(did, a_set) + set_flag(false, get_assoc(did, a_set));
    end_purge(did, a_set) =
      remove(did, a_set) + end_purge(get_assoc(did, a_set));
    trans_init(did, a_set) =
      remove(did, a_set) + trans_init(get_assoc(did, a_set));
    inc_purge(did, a_set) =
      remove(did, a_set) + inc_purge(get_assoc(did, a_set));
    dec_purge(did, a_set) =
      remove(did, a_set) + dec_purge(get_assoc(did, a_set));
    detach(did, a_set) = remove(did, a_set) + detach(get_assoc(did, a_set));
    transfer(flag, did, a_set) =
      remove(did, a_set) + set_flag(flag, detach(get_assoc(did, a_set)));
    re_attach(sp, did, a_set) =
      remove(did, a_set) + re_attach(sp, get_assoc(did, a_set));

    is_detached(did, a_set) and has_attached_branch(brid, a_set) =>
(* remove any detached entries and keep a new attached entry  *)
      prune_branch_entries(brid, did, a_set) =
        remove_branch_entries(brid, a_set) + get_assoc(get_attached_branch(brid, a_set), a_set);
    not(is_detached(did, a_set) and has_attached_branch(brid, a_set)) =>
(* keep old entry if it is attached or if there are no new attached entries  *)
      prune_branch_entries(brid, did, a_set) =
        remove_branch_entries(brid, a_set) + get_assoc(did, a_set);
    remove(empty_assocs, set) = set;
    remove(a_set + ent, set) = remove(a_set, remove(get_did(ent), set));
    chaining(empty_assocs) = empty_assocs;
    is_chaining(ent) => chaining(a_set + ent) = chaining(a_set) + ent;
    not(is_chaining(ent)) => chaining(a_set + ent) = chaining(a_set);
    chaining_subordinates(a_set) = chaining(subordinates(a_set));
    IsTwoWayChannel(empty_assocs) = empty_assocs;
    not(is_two_way(get_cu(ent))) =>
      IsTwoWayChannel(a_set + ent) = IsTwoWayChannel(a_set);
    is_two_way(get_cu(ent)) =>
      IsTwoWayChannel(a_set + ent) = IsTwoWayChannel(a_set) + ent;
    attached(empty_assocs) = empty_assocs;
    not(attached(ent)) => attached(a_set + ent) = attached(a_set);
    attached(ent) => attached(a_set + ent) = attached(a_set) + ent;
    remove_commitSPs(empty_assocs) = empty_assocs;
    flag(ent) =>
      remove_commitSPs(a_set + ent) = remove_commitSPs(a_set) +
        set_flag(chained_selected(get_sfu(ent)), remove_commitSPs(ent));
    not(flag(ent)) =>
      remove_commitSPs(a_set + ent) = remove_commitSPs(a_set) + ent;
    init_Assocs(a_set) = remove_commitSPs(attached(a_set));
    dialogues(empty_assocs) = empty_assocs;
    is_dialogue(ent) => dialogues(a_set + ent) = dialogues(a_set) + ent;
    not(is_dialogue(ent)) => dialogues(a_set + ent) = dialogues(a_set);
    channels(a_set) = remove(dialogues(a_set), a_set);
    branches(empty_assocs) = empty_assocs;
    is_dialogue(ent) and attached(ent) and not(flag(ent)) =>
```

```
(*  this is an attached dialogue with coordination-level none  *)
      branches(a_set + ent) = branches(a_set);
    is_channel(ent) or not(attached(ent)) or flag(ent) =>
      branches(a_set + ent) = branches(a_set) + ent;
    subordinates(empty_assocs) = empty_assocs;
    is_subordinate(ent) =>
      subordinates(a_set + ent) = subordinates(a_set) + ent;
    not(is_subordinate(ent)) => subordinates(a_set + ent) = subordinates(a_set);
    polarized_selected(empty_assocs) = empty_assocs;
    is_dialogue(ent) and polarized_selected(get_sfu(ent)) =>
      polarized_selected(a_set + ent) = polarized_selected(a_set) + ent;
    is_channel(ent) or not(polarized_selected(get_sfu(ent))) =>
      polarized_selected(a_set + ent) = polarized_selected(a_set);
    shared_selected(a_set) =
      remove(polarized_selected(a_set), dialogues(a_set));
    unchained_selected(empty_assocs) = empty_assocs;
    is_dialogue(ent) and unchained_selected(get_sfu(ent)) =>
      unchained_selected(a_set + ent) = unchained_selected(a_set) + ent;
    is_channel(ent) or not(unchained_selected(get_sfu(ent))) =>
      unchained_selected(a_set + ent) = unchained_selected(a_set);
    HasSentRcvd(x, empty_assocs) = empty_assocs;
    is_in_SentRcvd(x, ent) =>
      HasSentRcvd(x, a_set + ent) = HasSentRcvd(x, a_set) + ent;
    not(is_in_SentRcvd(x, ent)) =>
      HasSentRcvd(x, a_set + ent) = HasSentRcvd(x, a_set);
    HasPending(x, empty_assocs) = empty_assocs;
    is_in_Pending(x, ent) =>
      HasPending(x, a_set + ent) = HasPending(x, a_set) + ent;
    not(is_in_Pending(x, ent)) =>
      HasPending(x, a_set + ent) = HasPending(x, a_set);
    NotSentRcvd(x, a_set) = remove(HasSentRcvd(x, a_set), a_set);
    NoPending(x, a_set) = remove(NoPending(x, a_set), a_set);
  ofsort assoc_entry
    set_brid(brid,
      assoc(did, aet0, rch0, brid0, sent, pend, init_sp, purge, flag, attach)) =
      assoc(did, aet0, rch0, Opt(brid), sent, pend, init_sp, purge, flag, attach);
    add_SentRcvd(sp,
      assoc(did, aet0, rch0, brid0, sent, pend, init_sp, purge, flag, attach)) =
      assoc(did, aet0, rch0, brid0, sent + sp, pend, init_sp, purge, flag, attach);
    change_pending_to_sent(x, assoc(did, aet0, rch0, brid0, sent, pend, init_sp, purge, flag, attach)) =
      assoc(did,aet0,rch0,brid0,sent+get(x,pend),pend-x,init_sp,purge,flag,attach);
    add_Pending(sp,assoc(did, aet0, rch0, brid0, sent, pend, init_sp, purge, flag, attach)) =
      assoc(did, aet0, rch0, brid0, sent, pend + sp, init_sp, purge, flag, attach);
    remove_SentRcvd(x,assoc(did,aet0,rch0,brid0,sent,pend,init_sp,purge,flag,attach)) =
      assoc(did, aet0, rch0, brid0, sent - x, pend, init_sp, purge, flag, attach);
    remove_Pending(x, assoc(did,aet0,rch0,brid0,sent,pend,init_sp,purge,flag,attach)) =
      assoc(did, aet0, rch0, brid0, sent, pend - x, init_sp, purge, flag, attach);
    end_purge(assoc(did, aet0, rch0, brid0, sent, pend, init_sp, purge, flag, attach)) =
      assoc(did, aet0, rch0, brid0, sent, pend, init_sp, end(purge), flag, attach);
    remove_commitSPs(assoc(did, aet0, rch0, brid0, sent, pend, init_sp, purge, flag, attach)) =
      assoc(did,aet0,rch0,brid0,remove(commitSPs, sent),remove(commitSPs, pend),init_sp,purge,flag,attach);
    trans_init(assoc(did, aet0, rch0, brid0, sent, pend, init_sp, purge, flag, attach)) =
      assoc(did,aet0,rch0,brid0,sent + TPBeginTransactionReq,pend,init_sp,trans_init(purge),flag,attach);
    inc_purge(assoc(did, aet0, rch0, brid0, sent, pend, init_sp, purge, flag, attach)) =
      assoc(did, aet0, rch0, brid0, sent, pend, init_sp, inc(purge), flag, attach);
    dec_purge(assoc(did, aet0, rch0, brid0, sent, pend, init_sp, purge, flag, attach)) =
      assoc(did, aet0, rch0, brid0, sent, pend, init_sp, dec(purge), flag, attach);
    detach(assoc(did, aet0, rch0, brid0, sent, pend, init_sp, purge, flag, attach)) =
      assoc(did, aet0, rch0, brid0, sent, pend, init_sp, purge, flag, false);
    set_flag(new,assoc(did, aet0, rch0, brid0, sent, pend, init_sp, purge, flag, attach)) =
      assoc(did, aet0, rch0, brid0, sent, pend, init_sp, purge, new, attach);
    dialogue_entry(did, addr, rch0, init_sp) =
      assoc(did,get_aet0(addr),rch0,brid_absent,nilSP,nilSP,init_sp,init_purge(get_sfu(init_sp)),false,true);
    channel_entry(did, addr, init_sp) =
      assoc(did,get_aet0(addr),rch_absent,brid_absent,nilSP,nilSP,init_sp,purge_absent,false,true);
    recovery_entry(did, brid, sp) =
      assoc(did,aet_absent,rch_absent,Opt(brid),nilSP + sp,nilSP,sp,purge_absent,false,true);
    re_attach(sp,assoc(did, aet0, rch0, brid0, sent, pend, init_sp, purge, flag, attach)) =
      assoc(did, aet0, rch0, brid0, nilSP + sp, nilSP, init_sp, purge, false, true);
  ofsort set_of_dids
    get_dids(empty_assocs) = empty_dids;
    get_dids(a_set + ent) = get_dids(a_set) + get_did(ent);
  ofsort dc_id
    next_chid(empty_assocs) = first_chan;
    get_did(ent) lt next_chid(a_set) => next_chid(a_set + ent) = next_chid(a_set);
    not(get_did(ent) lt next_chid(a_set)) =>
      next_chid(a_set + ent) = inc(get_did(ent));
  ofsort purge_sort
      forall n, m : nat, b : bool, fu : functional_units
    inc(purging(n, m)) = purging(succ(n), m);
    inc(purging(b)) = purging(true);
    dec(purging(n, m)) = purging(decr(n), decr(m));
    dec(purging(b)) = purging(false);
    end(purging(n, m)) = purging(0, 0);
    end(purging(b)) = purging(false);
    trans_init(purging(n, m)) = purging(n, n);
    trans_init(purging(b)) = purging(b);
    shared_selected(fu) => init_purge(fu) = purging(0, 0);
    polarized_selected(fu) => init_purge(fu) = purging(false);
  ofsort bool
      forall n, m : nat, b : bool
    is_purging(purging(n, m)) = n ne 0;
    is_purging(purging(b)) = b;
    is_trans_init(purging(n, m)) = m ne 0;
    is_trans_init(purging(b)) = false;
endtype
(* ---------------------------------------------------------------------------
module association_info is
  dialogue_channel_identifiers, local_values, ServicePrim_Lists

/* purge_sort */
    type purge_sort is
        purging (N: nat,        /* shared  */
                 M: nat)        /* trans_init purging count */
    |   purging (B: bool)       /* polarized  */
    |   purge_absent
    endtype
    function init_purge (FU: functional_units) : purge_sort is
        if shared_selected (FU) then purging (0, 0)
        elsif polarized_selected (FU) then purging (false) endif
    endfunc
    function inc (purge: purge_sort) : purge_sort is
        case purge in
            purging(N:=n: nat, M:=m: nat) -> purging (succ (n), m)
        |   purging(B:=b: bool) -> purging (true)
        endcase
    endfunc
    function dec (purge: purge_sort) : purge_sort is
        case purge in
            purging(N:=n: nat, M:=m: nat) -> purging (decr (n), decr(m))
        |   purging(B:=b: bool) -> purging (false)
        endcase
    endfunc
    function end (purge: purge_sort) : purge_sort is
        case purge in
            purging(N:=n: nat, M:=m: nat) -> purging (0,0)
        |   purging(B:=b: bool) -> purging (false)
        endcase
    endfunc
    function is_purging (purge: purge_sort) : bool is endfunc
        case purge in
            purging(N:=n: nat, M:=m: nat) -> n ne 0
        |   purging(B:=b: bool) -> b
        endcase
    endfunc
    function is_trans_init (purge: purge_sort) : bool is endfunc
        case purge in
            purging(N:=n: nat, M:=m: nat) -> m ne 0
        |   purging(B:=b: bool) -> false
        endcase
    endfunc
    function trans_init (purge: purge_sort) : purge_sort is
        case purge in
            purging(N:=n: nat, M:=m: nat) -> purging (n,n)
        |   purging(B:=b: bool) -> purging (b)
        endcase
    endfunc

/* assoc_entry */
    type assoc_entry is
        assoc   (DID: dc_id,
                AETO: AE_title_Opt,
                RCHO: recovery_context_handle_Opt,
                BRIDO: branch_identifier_Opt,
                SENT: ServicePrimList, /*  sent list  */
                PEND: ServicePrimList, /*  pending list */
                INIT_SP: ServicePrim,
                PURGE: purge_sort,
                FLAG: bool,
                ATTACH: bool)
    endtype
    function get_did (E: assoc_entry) : dc_id is select DID in E endfunc
    function get_aet (E: assoc_entry) : AE_title is select AET in (select AETO in E) endfunc
    function get_SentRcvd (E: assoc_entry) : ServicePrimList is select SENT in E endfunc
    function get_Pending (E: assoc_entry) : ServicePrimList is select PEND in E endfunc
    function get_brid (E: assoc_entry) : branch_identifier is select BRID in (select BRIDO in E) endfunc
    function get_rchO (E: assoc_entry) : recovery_context_handle_Opt is select RCHO in E endfunc
    function get_sfu (E: assoc_entry) : functional_units is get_sfu (select INIT_SP in E) endfunc
    function get_bdc (E: assoc_entry) : tp_begin_dialogue_confirmation is get_bdc (select INIT_SP in E) endfunc
    function get_cu (E: assoc_entry) : channel_utilization is get_cu (select INIT_SP in E) endfunc
    function is_superior (BI: branch_identifier) : bool is not (is_subordinate (BI)) endfunc
    function is_subordinate (BI: branch_identifier) : bool is superiors_name(BI) match my_aet endfunc

    function attached (E: assoc_entry) : bool is select ATTACH in E endfunc
    function flag (E: assoc_entry) : bool is select FLAG in E endfunc
    function is_dialogue (E: assoc_entry) : bool is
        is_dial (get_did (E)) and attached (E) endfunc
    function is_channel (E: assoc_entry) : bool is
        is_chan (get_did (E)) and attached (E)  endfunc
    function has_brid (E: assoc_entry) : bool is
        (select BRIDO in E) match Opt (...) endfunc
    function is_superior (E: assoc_entry) : bool is
        has_brid (E) and is_superior (get_brid (E)) endfunc
    function is_subordinate (E: assoc_entry) : bool is
        has_brid (E) and is_subordinate (get_brid (E)) endfunc
    function is_purging (E: assoc_entry) : bool is
        is_purging (select PURGE in E) endfunc
    function is_trans_init_purging (E: assoc_entry) : bool is
        is_trans_init (select PURGE in E) endfunc
    function is_in_SentRcvd (K: LookUpKey, E: assoc_entry) : bool is
        K IsIn (select SENT in E) endfunc
    function is_in_Pending (K: LookUpKey, E: assoc_entry) : bool is
        K IsIn (select PEND in E) endfunc
    function is_free_channel (E: assoc_entry) : bool is
        attached (E) and not (is_in_SentRcvd (CAFPlease_requ, E))  endfunc
    function is_chaining (E: assoc_entry) : bool is
        attached (E) and chained_selected (get_sfu (E))
        and not (is_in_SentRcvd (DeferEndDialogue, E)) endfunc

/* set_of_assocs */
    type set_of_assocs is
        empty_assocs
    |   _+_  (SET: set_of_assocs, ENT: assoc_entry)
    endtype

    function get_sup (SET: set_of_assocs) : dc_id is
        case SET in
          empty_assocs -> did_absent
        | (SET:=S: set_of_assocs) + (ENT:=E: assoc_entry) when is_superior(E)
                -> get_did(E)
        | (SET:=S: set_of_assocs) + (ENT:=E: assoc_entry) -> get_sup(S)
        endcase
    endfunc
```

```
function add (ENT: assoc_entry, SET: set_of_assocs) : set_of_assocs is SET + ENT endfunc
function IsEmpty (SET: set_of_assocs) : bool is SET match empty_assocs endfunc
function match (D: dc_id, E: assoc_entry) : bool is D eq get_did (E) endfunc

function remove (D: dc_id, S: set_of_assocs) : set_of_assocsK is
    case S in
        empty_assocs -> empty_assocs
    |   (SET:=S1: set_of_assocs) + (ENT:=E: assoc_entry) ->
            if match (D, E) then S1
            else remove (D, S1) + E endif
    endcase
endfunc
function remove (S1: set_of_assocs, S2: set_of_assocs) : set_of_assocs is
    case S1 in
        empty_assocs -> S2
    |   (SET:=S: set_of_assocs) + (ENT:=E: assoc_entry) ->
            remove (S, remove (get_did (E), S2))
    endcase
endfunc

function get_assoc (DID: dc_id, A_SET: set_of_assocs) : assoc_entry is
    let (SET:=S: set_of_assocs) + (ENT:=E: assoc_entry) = A_SET in
        if match (DID, E) then E
        else get_assoc(DID, A_SET) endif
endfunc

function get_aet (DID: dc_id, A_SET: set_of_assocs) : AE_title is
    get_aet(get_assoc(DID, A_SET)) endfunc

function get_SentRcvd (D: dc_id, S: set_of_assocs) : ServicePrimList is
    get_SentRcvd (get_assoc (D, S)) endfunc
function get_SentRcvd (K: LookUpKey, D: dc_id, S: set_of_assocs) : ServicePrim is
    get (K, get_SentRcvd (D, S)) endfunc
function get_Pending (D: dc_id, S: set_of_assocs) : ServicePrimList is
    get_Pending (get_assoc (D, S)) endfunc
function get_Pending (K: LookUpKey, D: dc_id, S: set_of_assocs) : ServicePrim is
    get (K, get_Pending (D, S)) endfunc
function get_brid (D: dc_id, S: set_of_assocs) : branch_identifier is
    get_brid (get_assoc (D, S)) endfunc
function get_rchO (D: dc_id, S: set_of_assocs) : recovery_context_handle_Opt is
    get_rchO (get_assoc (D, S)) endfunc
function get_sfu (D: dc_id, S: set_of_assocs) : functional_units is
    get_sfu (get_assoc (D, S)) endfunc
function get_bdc (D: dc_id, S: set_of_assocs) : tp_begin_dialogue_confirmation is
    get_bdc (get_assoc (D, S)) endfunc
function get_cu (D: dc_id, S: set_of_assocs) : channel_utilization is
    get_cu (get_assoc (D, S)) endfunc

function is_did_in (D: dc_id, S: set_of_assocs) : bool is
    let (SET:=A_SET: set_of_assocs) + (ENT:=E: assoc_entry) = S in
        match (D, E) or is_did_in (D, A_SET) endfunc
function is_dial_in (D: dc_id, S: set_of_assocs) : bool is
    is_did_in (D, S) and is_dialogue (get_assoc (D, S)) endfunc
function is_chan_in (D: dc_id, S: set_of_assocs) : bool is
```

```
    is_did_in (D, S) and is_channel (get_assoc (D, S)) endfunc
function is_superior (D: dc_id, S: set_of_assocs) : bool is
    is_superior (get_assoc (D, S)) endfunc
function is_subordinate (D: dc_id, S: set_of_assocs) : bool is
    is_subordinate (get_assoc (D, S)) endfunc
function is_coord_commitment (D: dc_id, S: set_of_assocs) : bool is
    flag (get_assoc (D, S)) endfunc /* select FLAG in get_assoc (D, S) */
function is_coord_none (D: dc_id, S: set_of_assocs) : bool is
    not (is_coord_commitment (D, S)) endfunc
function is_detached (D: dc_id, S: set_of_assocs) : bool is
    not (is_did_in (D, S)) or not (attached (get_assoc (D, S))) endfunc
function is_purging (D: dc_id, S: set_of_assocs) : bool is
    is_purging (get_assoc (D, S)) endfunc
function is_trans_init_purging (D: dc_id, S: set_of_assocs) : bool is
    is_trans_init_purging (get_assoc (D, S)) endfunc
function transfered_to_another_tppm (D: dc_id, S: set_of_assocs) : bool is
   is_did_in (D, S) and not (attached (get_assoc (D, S))) and flag (get_assoc (D, S))  endfunc
function is_free_channel (D: dc_id, S: set_of_assocs) : bool is
    is_did_in (D, S) and is_free_channel(get_assoc (D, S)) endfunc
function retry (D: dc_id, S: set_of_assocs) : bool is
    is_did_in (D, S) and not(attached(get_assoc (D, S))) and flag(get_assoc (D, S)) endfunc
function is_chaining (D: dc_id, S: set_of_assocs) : bool is
    is_did_in (D, S) and is_chaining (get_assoc (D, S)) endfunc


function is_in_SentRcvd (K: LookUpKey, D: dc_id, S: set_of_assocs) : bool is
    K IsIn get_SentRcvd (D, S) endfunc
function is_in_Pending (K: LookUpKey, D: dc_id, S: set_of_assocs) : bool is
    K IsIn get_Pending (D, S) endfunc
function is_last (K: LookUpKey, D: dc_id, S: set_of_assocs) : bool is
    last (get_SentRcvd (D, S)) eq K endfunc
function get_attached_branch (BI: branch_identifier, S: set_of_assocs) : dc_id is
    case S in
        empty_assocs -> did_absent
    |   (SET:=S: set_of_assocs) + (ENT:=E: assoc_entry) ->
            if attached (E) and has_brid (E) and (get_brid (E) eq BI) then get_did (E)
            else get_attached_branch (BI, S) endif
    endcase
endfunc
function get_branch_entry (BI: branch_identifier, S: set_of_assocs) : dc_id is
    case S in
        empty_assocs -> did_absent
    |   (SET:=S: set_of_assocs) + (ENT:=E: assoc_entry) ->
            if has_brid (E) and (get_brid (E) eq BI) then get_did (E)
            else get_attached_branch (BI, S) endif
    endcase
endfunc
function has_attached_branch (BI: branch_identifier, S: set_of_assocs) : bool is
    not (is_absent (get_attached_branch (BI, S))) endfunc
function remove_branch_entries (BI: branch_identifier, S: set_of_assocs) : set_of_assocs is
    case S in
        empty_assocs -> empty_assocs
    |   (SET:=S: set_of_assocs) + (ENT:=E: assoc_entry) ->
            if has_brid (E) and (get_brid (E) eq BI) then remove_branch_entries (BI, S)
            else remove_branch_entries (BI, S) + E endif
    endcase
endfunc
function prune_branch_entries (BI: branch_identifier, D: dc_id, S: set_of_assocs) : set_of_assocs is
    if is_detached (D, S) and has_attached_branch (BI, S) then
        remove_branch_entries (BI, S) + get_assoc (get_attached_branch (BI, S), S)
    else remove_branch_entries (BI, S) +  get_assoc (D, S) endif
endfunc
function set_brid (BI: branch_identifier, D: dc_id, S: set_of_assocs) : set_of_assocs is
    remove (D, S) + set_brid (BI, get_assoc (D, S)) endfunc
function remove_SentRcvd (K: LookUpKey, D: dc_id, S: set_of_assocs) : set_of_assocs is
    remove (D, S) + remove_SentRcvd (K, get_assoc (D, S)) endfunc
function remove_Pending (K: LookUpKey, D: dc_id, S: set_of_assocs) : set_of_assocs is
    remove (D, S) + remove_Pending (x, get_assoc(D, S)) endfunc
function change_pending_to_sent (K: LookUpKey, D: dc_id, S: set_of_assocs) : set_of_assocs is
    remove (D, S) + change_pending_to_sent (K, get_assoc (D, S)) endfunc
function add_SentRcvd (SP: ServicePrim, D: dc_id, S: set_of_assocs) : set_of_assocs is
    remove (D, S) + add_SentRcvd (SP, get_assoc (D, S)) endfunc
function add_Pending (SP: ServicePrim, D: dc_id, S: set_of_assocs) : set_of_assocs is
    remove (D, S) + add_Pending (SP, get_assoc (D, S)) endfunc
function add_SentRcvd (SP: ServicePrim, S1: set_of_assocs, S2: set_of_assocs) : set_of_assocs is
    case S1 in
        empty_assocs -> S2
         |   (SET:=S: set_of_assocs) + (ENT:=E: assoc_entry)
               -> add_SentRcvd (SP, S, add_SentRcvd (SP, get_did (E), S2))
    endcase
endfunc

function detach (D: dc_id, S: set_of_assocs) : set_of_assocs is
    remove (D, S) + detach (get_assoc(D, S)) endfunc
function coord_commitment (D: dc_id, S: set_of_assocs) : set_of_assocs is
    remove (D, S) + set_flag (true, get_assoc (D, S)) endfunc
function coord_none (D: dc_id, S: set_of_assocs) : set_of_assocs is
    remove (D, S) + set_flag (false, get_assoc (D, S)) endfunc
function retry_needed (D: dc_id, S: set_of_assocs) : set_of_assocs is
    remove (D, S) + set_flag (true, get_assoc (D, S)) endfunc
function retry_sent (D: dc_id, S: set_of_assocs) : set_of_assocs is
    remove (D, S) + set_flag (false, get_assoc (D, S)) endfunc
function end_purge (D: dc_id, S: set_of_assocs) : set_of_assocs is
    remove (D, S) + end_purge (get_assoc (D, S)) endfunc
function trans_init (D: dc_id, S: set_of_assocs) : set_of_assocs is
    remove (D, S) + trans_init (get_assoc (D, S)) endfunc
function inc_purge (D: dc_id, S: set_of_assocs) : set_of_assocs is
    remove (D, S) + inc_purge (get_assoc (D, S)) endfunc
function dec_purge (D: dc_id, S: set_of_assocs) : set_of_assocs is
    remove (D, S) + dec_purge (get_assoc (D, S)) endfunc
function transfer (F: bool, D: dc_id, S: set_of_assocs) : set_of_assocs is
    remove (D, S) + set_flag (F, detach (get_assoc (D, S))) endfunc
function re_attach (SP: ServicePrim, D: dc_id, S: set_of_assocs) : set_of_assocs is
    remove (D, S) + re_attach (SP, get_assoc (D, S)) endfunc

function set_brid (BI: branch_identifier, E: assoc_entry) : assoc_entry is
    E.{BRIDO:= Opt (BI)} endfunc
function remove_SentRcvd (K: LookUpKey, E: assoc_entry) : assoc_entry is
```

```
        E.{SENT:= (select SENT in E) - K} endfunc
    function remove_Pending (K: LookUpKey, E: assoc_entry) : assoc_entry is
        E.{PEND:= (select PEND in E) - K} endfunc
    function change_pending_to_sent (K: LookUpKey, E: assoc_entry) : assoc_entry is
        E.{SENT:= (select SENT in E) + get (K, (select PEND IN E)),
           PEND:= (select PEND in E) - x} endfunc
    function re_attach (SP: ServicePrim, E: assoc_entry) : assoc_entry is
        E.{SENT:= nilSP + SP, PEND:= nilSP} endfunc
    function add_SentRcvd (SP: ServicePrim, E: assoc_entry) : assoc_entry is
        E.{SENT:= (select SENT in E) + SP} endfunc
    function add_Pending (SP: ServicePrim, E: assoc_entry) : assoc_entry is
        E.{PEND:= (select PEND in E) + SP} endfunc
    function set_flag (B: bool, E: assoc_entry) : assoc_entry is
        E.{FLAG:=B} endfunc
    function detach (E: assoc_entry) : assoc_entry is
        E.{ATTACH:= false} endfunc
    function remove_commitSPs (E: assoc_entry) : assoc_entry is
        E.{SENT:= remove (commitSP, (select SENT in E)),
           PEND:= remove (commitSP, (select PEND in E))} endfunc
    function end_purge (E: assoc_entry) : assoc_entry is
        E.{PURGE:= end (select PURGE in E)} endfunc
    function trans_init (E: assoc_entry) : assoc_entry is
        E.{SENT:= (select SENT in E) + TPBeginTransactionReq,
           PURGE:= trans_init (select PURGE in E)} endfunc
    function inc_purge (E: assoc_entry) : assoc_entry is
        E.{PURGE:= inc (select PURGE in E)} endfunc
    function dec_purge (E: assoc_entry) : assoc_entry is
        E.{PURGE:= dec (select PURGE in E)}  endfunc
    function dialogue_entry (D: dc_id, A: address,
        R: recovery_context_handle_Opt, SP: ServicePrim) : assoc_entry is
        assoc(D,get_aetO(A),R,brid_absent,nilSP,nilSP,SP,init_purge(get_sfu(SP)),false,true) endfunc
    function channel_entry (D: dc_id, A: address, SP: ServicePrim) : assoc_entry is
        assoc(D,get_aetO(A),rch_absent,brid_absent,nilSP,nilSP,SP,purge_absent,false,true) endfunc
    function recovery_entry (D: dc_id, BI: branch_identifier, SP: ServicePrim) : assoc_entry is
        assoc(D,aet_absent,rch_absent,Opt(BI),nilSP+SP,nilSP,SP,purge_absent,false,true) endfunc
    function has_sup (S: set_of_assocs) : bool is not (is_absent (get_sup (S))) endfunc
    function is_root (S: set_of_assocs) : bool is not (has_sup (S))endfunc
    function is_leaf (S: set_of_assocs) : bool is IsEmpty (subordinates (S)) endfunc
    function is_root_or_intermediate (S: set_of_assocs) : bool is not (is_leaf (S)) endfunc
    function is_intermediate_or_leaf (S: set_of_assocs) : bool is has_sup (S) endfunc
    function is_intermediate (S: set_of_assocs) : bool is
        is_root_or_intermediate (S) and is_intermediate_or_leaf (S) endfunc
    function any_detached (S: set_of_assocs) : bool is
        not (IsEmpty (remove (attached (S), S)))endfunc
    function any_chaining (S: set_of_assocs) : bool is
        not (IsEmpty (chaining (S))) endfunc
    function rollback_rep_compl (S: set_of_assocs) : bool is
        not (has_sup (S)) or is_in_SentRcvd (RollbackRsp, sup_dial, S) or
        is_in_SentRcvd (RollbackCnf, sup_dial, S) endfunc
    function init_Assocs (S: set_of_assocs) : set_of_assocs is
        remove_commitSPs (attached (S)) endfunc
    function dialogues (S: set_of_assocs) : set_of_assocs is
        case S in
            empty_assocs -> empty_assocs
        |   (SET:=S1: set_of_assocs) + (ENT:=E: assoc_entry) ->
                if is_dialogue (E) then dialogues (S1) + ent
                else dialogues (S1) endif
        endcase
    endfunc
    function channels (S: set_of_assocs) : set_of_assocs is
        remove (dialogues (S), S) endfunc
    function branches (S: set_of_assocs) : set_of_assocs is
            empty_assocs -> empty_assocs
        |   (SET:=S1: set_of_assocs) + (ENT:=E: assoc_entry) ->
                if is_dialogue (ent) and attached (ent) and not (flag (E)) then branches (S1)
                elsif is_channel(E) or not(attached(E)) or flag(E) then
                     branches(S1) + E endif
        endcase
    endfunc
    function subordinates (S: set_of_assocs) : set_of_assocs is
        case S in
            empty_assocs -> empty_assocs
        |   (SET:=S1: set_of_assocs) + (ENT:=E: assoc_entry) ->
                if is_subordinate (E) then subordinates (S1) + E
                else subordinates (S1) endif
        endcase
    endfunc
    function attached (S: set_of_assocs) : set_of_assocs is
        case S in
            empty_assocs -> empty_assocs
        |   (SET:=S1: set_of_assocs) + (ENT:=E: assoc_entry) ->
                if attached (E) then attached (S1) + E
                else attached (S1) endif
        endcase
    endfunc
    function remove_commitSPs (S: set_of_assocs) : set_of_assocs is
        case S in
            empty_assocs -> empty_assocs
        |   (SET:=S1: set_of_assocs) + (ENT:=E: assoc_entry) ->
                if flag (E) then remove_commitSPs (S1) +
                        set_flag (chained_selected (get_sfu (E)), remove_commitSPs (E))
                else remove_commitSPs (S1) + E endif
        endcase
    endfunc
    function IsTwoWayChannel (S: set_of_assocs) : set_of_assocs is
        case S in
            empty_assocs -> empty_assocs
        |   (SET:=S1: set_of_assocs) + (ENT:=E: assoc_entry) ->
                if is_two_way (get_cu (E)) then IsTwoWayChannel (S1) + E
                else IsTwoWayChannel (S1) endif
        endcase
    endfunc
    function chaining (S: set_of_assocs) : set_of_assocs is
        case S in
            empty_assocs -> empty_assocs
        |   (SET:=S1: set_of_assocs) + (ENT:=E: assoc_entry) ->
                if is_chaining (E) then chaining (S1) + E
                else chaining (S1) endif
        endcase
    endfunc
    function chaining_subordinates (S: set_of_assocs) : set_of_assocs is
        chaining (subordinates (S1)) endfunc
    function polarized_selected (S: set_of_assocs) : set_of_assocs is
        case S in
            empty_assocs -> empty_assocs
        |   (SET:=S1: set_of_assocs) + (ENT:=E: assoc_entry) ->
                if is_dialogue (E) and polarized_selected (get_sfu (E)) then
                        polarized_selected(S) + E
                else polarized_selected (S1) endif
        endcase
    endfunc
    function shared_selected (S: set_of_assocs) : set_of_assocs is
        remove (polarized_selected (S1), dialogues (S1)) endfunc
    function unchained_selected (S: set_of_assocs) : set_of_assocs is
        case S in
            empty_assocs -> empty_assocs
        |   (SET:=S1: set_of_assocs) + (ENT:=E: assoc_entry) ->
                if is_dialogue (E) and unchained_selected (get_sfu (E))
                then unchained_selected (S1) + E
                elseunchained_selected (S1) endif
        endcase
    endfunc
    function HasSentRcvd (K: LookUpKey, S: set_of_assocs) : set_of_assocs is
        case S in
            empty_assocs -> empty_assocs
        |   (SET:=S1: set_of_assocs) + (ENT:=E: assoc_entry) ->
                if is_in_SentRcvd (K, E) then HasSentRcvd (K, S) + E
                else HasSentRcvd (K, S) endif
        endcase
    endfunc
    function HasPending (K: LookUpKey, S: set_of_assocs) : set_of_assocs is
        case S in
            empty_assocs -> empty_assocs
        |   (SET:=S1: set_of_assocs) + (ENT:=E: assoc_entry) ->
                if is_in_Pending (K, E) then HasPending (K, S) + E
                else HasPending (K, S) endif
        endcase
    endfunc
    function NotSentRcvd (K: LookUpKey, S: set_of_assocs) : set_of_assocs is
        remove (HasSentRcvd (K, S), S) endfunc
    function NoPending (K: LookUpKey, S: set_of_assocs) : set_of_assocs is
        remove (NoPending (K, S), S) endfunc
    function get_dids (S: set_of_assocs) : set_of_dids is
        case S in
            empty_assocs -> empty_dids
        |   (SET:=S1: set_of_assocs) + (ENT:=E: assoc_entry) ->
                get_dids (S1) + get_did (E)
        endcase
    endfunc
    function next_chid (S: set_of_assocs) : dc_id is
        case S in
            empty_assocs -> first_chan
        |   (SET:=S1: set_of_assocs) + (ENT:=E: assoc_entry) ->
                if get_did (E) lt next_chid (S1) then next_chid (S1)
                else inc (get_did (E)) endif
        endcase
    endfunc

endmod
============================================================================ *)

type match_channel_CAFPleaseReq is association_info
opns
  match_channel : AE_Title, ServicePrim -> bool
  match_channel : address, ServicePrim -> bool
  match_channel : AE_Title_Opt, ServicePrim -> bool
  eq_CAFPleaseReq : ServicePrim, ServicePrim -> bool
  CAFPleaseReq_in : ServicePrim, ServicePrimList -> bool
  has_matching_channel : ServicePrim, set_of_assocs -> bool
eqns
    forall
      addr : address,
      aet : AE_Title,
      sp, sp1 : ServicePrim,
      l : ServicePrimList,
      a_set : set_of_assocs,
      ent : assoc_entry
  ofsort bool
    match_channel(aet, sp) = IsCAFPleaseReq(sp) and (aet eq get_aet(sp));
    match_channel(addr, sp) = match_channel(get_aetO(addr), sp);
    match_channel(aet_absent, sp) = false;
    match_channel(Opt(aet), sp) = match_channel(aet, sp);
    eq_CAFPleaseReq(sp, sp1) =
      IsCAFPleaseReq(sp)
      and
      IsCAFPleaseReq(sp1)
      and
      (get_aet(sp) eq get_aet(sp1));
    CAFPleaseReq_in(sp, nilSP) = false;
    CAFPleaseReq_in(sp, l + sp1) =
      eq_CAFPleaseReq(sp, sp1) or CAFPleaseReq_in(sp, l);
    has_matching_channel(sp, empty_assocs) = false;
    has_matching_channel(sp, a_set + ent) =
      (is_free_channel(ent) and match_channel(get_aet(ent), sp))
      or
      has_matching_channel(sp, a_set)
endtype
(* -------------------------------------------------------------------------

module match_channel_CAFPleaseReq is association_info
    function match_channel (AET: AE_Title, SP: ServicePrim) : bool is
        IsCAFPleaseReq (SP) and (AET eq get_aet (SP)) endfunc
    function match_channel (A: address, SP: ServicePrim) : bool is
        match_channel (get_aetO (A), SP) endfunc
    function match_channel(AETO: AE_Title_Opt, SP: ServicePrim) : bool is
        case AETO in
```

```
            aet_absent -> false
        |   Opt (AET:=AET: AE_Title) -> match_channel (AET, SP)
        endcase
    endfunc
    function eq_CAFPleaseReq (SP1: ServicePrim, SP2: ServicePrim) : bool is
      IsCAFPleaseReq(SP1) and IsCAFPleaseReq(SP2) and
      (get_aet (SP1) eq get_aet (SP2)) endfunc
    function CAFPleaseReq_in (SP: ServicePrim, L: ServicePrimList) : bool is
        case L in
            nilSP -> false
        |   (SPL:=SPL: ServicePrimList) + (SP:=SP1: ServicePrim) ->
                eq_CAFPleaseReq (SP, SP1) or CAFPleaseReq_in (SP, SPL)
        endcase
    endfunc
    function has_matching_channel (SP: ServicePrim, S: set_of_assocs) : bool is
        case S in
            empty_assocs -> false
        |   (SET:=A_SET: set_of_assocs) + (ENT:=E: assoc_entry) ->
        (is_free_channel (E) and match_channel (get_aet (E), SP))
        or has_matching_channel (SP, A_SET)
endmod
============================================================================ *)

type tppm_info is log_records, match_channel_CAFPleaseReq
sorts tppm_info, tppm_state
opns
(*tppm_info*)
  tppm_info (*! constructor *) :
    tppm_state,
    atomic_action_identifier,
    ServicePrimList, (* sent list *)
    ServicePrimList, (* pending list*)
    bool (*  tp_done_owing  *)  -> tppm_info
  empty_tppm_info (*! constructor *) : -> tppm_info
  change_state_active : atomic_action_identifier -> tppm_info
  get_aaid : tppm_info -> atomic_action_identifier
  TPDone_owing : tppm_info -> tppm_info
  TPDone_sent : tppm_info -> tppm_info
  add_SentRcvd : ServicePrim, tppm_info -> tppm_info
  change_pending_to_sent : LookUpKey, tppm_info -> tppm_info
  add_Pending : ServicePrim, tppm_info -> tppm_info
  remove_SentRcvd, remove_Pending : LookUpKey, tppm_info -> tppm_info
  get_SentRcvd, get_Pending : tppm_info -> ServicePrimList
  get_SentRcvd, get_Pending : LookUpKey, tppm_info -> ServicePrim
  is_in_SentRcvd, is_in_Pending : LookUpKey, tppm_info -> bool
  OwesTPDone : tppm_info -> bool (*tp_done_owing*)
  log_ready_record, log_commit_record : set_of_assocs, tppm_info -> log_record
  recreate_tppm : log_record -> tppm_info

(* tppm_state *)
  ready (*! constructor *),
  active (*! constructor *),
  none (*! constructor *),
  dec_cmt (*! constructor *),
  dec_rbk (*! constructor *) : -> tppm_state
  is_ready, is_active, is_none, is_dec_cmt, is_dec_rbk : tppm_state -> bool
  tppm_nat : tppm_state -> nat
  get_state : tppm_info -> tppm_state
  change_state : tppm_state, tppm_info -> tppm_info
eqns
    forall
      tppm : tppm_info,
      ts, ts1 : tppm_state,
      aaid : atomic_action_identifier,
      brid : branch_identifier,
      sent, pend : ServicePrimList,
      tdo : bool  (*tp_done_owing*),
      log : log_record,
      a_set : set_of_assocs,
      sp : ServicePrim,
      x : LookUpKey,
      chid, did : dc_id,
      aetO : AE_title_Opt
  ofsort tppm_info
    change_state_active(aaid) = tppm_info(active, aaid, nilSP, nilSP, false);
    change_state(ts1, tppm_info(ts, aaid, sent, pend, tdo)) =
      tppm_info(ts1, aaid, sent, pend, is_dec_cmt(ts1) or is_dec_rbk(ts1));
    TPDone_owing(tppm_info(ts, aaid, sent, pend, tdo)) =
      tppm_info(ts, aaid, sent, pend, true);
    TPDone_sent(tppm_info(ts, aaid, sent, pend, tdo)) =
      tppm_info(ts, aaid, sent, pend, false);
    add_SentRcvd(sp, tppm_info(ts, aaid, sent, pend, tdo)) =
      tppm_info(ts, aaid, sent + sp, pend, tdo);
    change_pending_to_sent(x, tppm_info(ts, aaid, sent, pend, tdo)) =
      tppm_info(ts, aaid, sent + get(x, pend), pend - x, tdo);
    add_Pending(sp, tppm_info(ts, aaid, sent, pend, tdo)) =
      tppm_info(ts, aaid, sent, pend + sp, tdo);
    remove_SentRcvd(x, tppm_info(ts, aaid, sent, pend, tdo)) =
      tppm_info(ts, aaid, sent - x, pend, tdo);
    remove_Pending(x, tppm_info(ts, aaid, sent, pend, tdo)) =
      tppm_info(ts, aaid, sent, pend - x, tdo);
    is_ready(get_state(log)) =>
      recreate_tppm(log) = tppm_info(ready, get_aaid(log), nilSP, nilSP, false);
    is_commit(get_state(log)) =>
      recreate_tppm(log) =
        tppm_info(dec_cmt, get_aaid(log), nilSP, nilSP, true);
  ofsort log_record
    log_ready_record(a_set, tppm) =
      log_record(ready, get_aaid(tppm), branch_list(a_set));
    log_commit_record(a_set, tppm) =
      log_record(commit, get_aaid(tppm), branch_list(a_set));
  ofsort ServicePrimList
    get_SentRcvd(tppm_info(ts, aaid, sent, pend, tdo)) = sent;
    get_Pending(tppm_info(ts, aaid, sent, pend, tdo)) = pend;
  ofsort ServicePrim
    get_SentRcvd(x, tppm) = get(x, get_SentRcvd(tppm));
    get_Pending(x, tppm) = get(x, get_Pending(tppm));
  ofsort tppm_state
```

```
    get_state(tppm_info(ts, aaid, sent, pend, tdo)) = ts;
    get_state(empty_tppm_info) = none;
  ofsort atomic_action_identifier
    get_aaid(tppm_info(ts, aaid, sent, pend, tdo)) = aaid;
  ofsort bool
    is_in_SentRcvd(x, tppm) = x IsIn get_SentRcvd(tppm);
    is_in_Pending(x, tppm) = x IsIn get_Pending(tppm);
    OwesTPDone(tppm_info(ts, aaid, sent, pend, tdo)) = tdo;
    is_none(ts) = tppm_nat(ts) eq 0;
    is_active(ts) = tppm_nat(ts) eq 1;
    is_dec_cmt(ts) = tppm_nat(ts) eq 2;
    is_dec_rbk(ts) = tppm_nat(ts) eq 3;
    is_ready(ts) = tppm_nat(ts) eq 4;
  ofsort nat
    tppm_nat(none) = 0;
    tppm_nat(active) = 1;
    tppm_nat(dec_cmt) = 2;
    tppm_nat(dec_rbk) = 3;
    tppm_nat(ready) = 4;
endtype
(* ---------------------------------------------------------------------------
module tppm_info is log_records, match_channel_CAFPleaseReq

/* tppm_state */
    type tppm_state is
        ready
    |   active
    |   none
    |   dec_cmt
    |   dec_rbk
    endtype
    function is_none (TS: tppm_state) is TS match none endfunc
    function is_active (TS: tppm_state) is TS match active endfunc
    function is_dec_cmt (TS: tppm_state) is TS match dec_cmt endfunc
    function is_dec_rbk (TS: tppm_state) is TS match dec_rbk endfunc
    function is_ready (TS: tppm_state) is TS match ready endfunc
/* no translation is needed for function tppm_nat */

/* tppm_info */
    type tppm_info is
        empty_tppm_info
    |   tppm_info (TS: tppm_state,
                AAID: atomic_action_identifier,
                SENT: ServicePrimList,
                PEND: ServicePrimList,
                TDO: bool)
    endtype

    function change_state_active (A: atomic_action_identifier) : tppm_info is
        tppm_info (active, A, nilSP, nilSP, false) endfunc
    function change_state (TS1: tppm_state, TPPM: tppm_info) : tppm_info is
        TPPM.{TS:=TS1, TDO:= is_dec_cmt (TS1) or is_dec_rbk (TS1)) endfunc
    function TPDone_owing (TPPM: tppm_info) : tppm_info is
        TPPM.{TDO:= true} endfunc
    function TPDone_sent (TPPM: tppm_info) : tppm_info is
        TPPM.{TDO:= false} endfunc
    function add_SentRcvd (SP: ServicePrim, TPPM: tppm_info) : tppm_info is
        TPPM.{SENT:= (select SENT in TPPM) + sp} endfunc
    function change_pending_to_sent (K: LookUpKey, TPPM: tppm_info) : tppm_info is
        TPPM.{SENT:= (select SENT in TPPM) + get (K, (select PEND in TPPM)),
              PEND:= (select PEND in TPPM) - K} endfunc
    function add_Pending (SP: ServicePrim, TPPM: tppm_info) : tppm_info is
        TPPM.{PEND:= (select PEND in TPPM) + SP} endfunc
    function remove_SentRcvd (K: LookUpKey, TPPM: tppm_info) : tppm_info is
        TPPM.{SENT:= (select SENT in TPPM) - K} endfunc
    function remove_Pending (K: LookUpKey, TPPM: tppm_info) : tppm_info is
        TPPM.{PEND:= (select PEND in TPPM) - K} endfunc
    function recreate_tppm (LOG: log_record) : tppm_info is
        if is_ready (get_state (LOG)) then
            tppm_info (ready, get_aaid (LOG), nilSP, nilSP, false)
        else tppm_info (dec_cmt, get_aaid (LOG), nilSP, nilSP, true) endif
    endfunc
    function log_ready_record (SET: set_of_assocs, TPPM: tppm_info) : log_record is
        log_record (ready, get_aaid (TPPM), branch_list (SET)) endfunc
    function log_commit_record (SET: set_of_assocs, TPPM: tppm_info) : log_record is
        log_record (commit, get_aaid (TPPM), branch_list (SET)) endfunc
    function get_SentRcvd (TPPM: tppm_info) : ServicePrimList is select SENT in TPPM endfunc
    function get_Pending(TPPM: tppm_info) : ServicePrimList is select PEND in TPPM endfunc
    function get_SentRcvd (K: LookUpKey, TPPM: tppm_info) : ServicePrim is
        get (K, get_SentRcvd (TPPM)) endfunc
    function get_Pending (K: LookUpKey, TPPM: tppm_info) : ServicePrim is
        get (K, get_Pending (TPPM)) endfunc
    function get_state (TPPM: tppm_info) : tppm_state is
        case TPPM in
            tppm_info (...) -> select TS in TPPM
        |   empty_tppm_info -> none
        endcase
    endfunc
    function get_aaid (TPPM: tppm_info) : atomic_action_identifier is select AAID in TPPM endfunc
    function is_in_SentRcvd (K: LookUpKey, TPPM: tppm_info) : bool is K IsIn get_SentRcvd (TPPM) endfunc
    function is_in_Pending (K: LookUpKey, TPPM: tppm_info) : bool is K IsIn get_Pending (TPPM) endfunc
    function OwesTPDone (TPPM: tppm_info) : bool is select TDO in TPPM endfunc
endmod
========================================================================= *)

type last_ready_commit_rollback_received is TPPM_info
opns
  last_ready_received, last_commit_received, last_rollback_received :
    set_of_assocs, tppm_info -> bool
  ready_received, commit_received, rollback_received : set_of_assocs -> bool
eqns
    forall tppm : tppm_info, a_set : set_of_assocs, ent : assoc_entry
  ofsort bool
    last_ready_received(a_set, tppm) =
      is_in_SentRcvd(TPCommit_requ, tppm) and ready_received(a_set);
    last_commit_received(a_set, tppm) =
      not(OwesTPDone(tppm)) and commit_received(a_set);
    last_rollback_received(a_set, tppm) =
      not(OwesTPDone(tppm)) and rollback_received(a_set);
    ready_received(empty_assocs) = true;
    ready_received(a_set + ent) =
      ( is_in_SentRcvd(AFPrepare_requ, ent)
        implies
        is_in_SentRcvd(CReady_indi, ent)
      )
      and
      ready_received(a_set);
    commit_received(empty_assocs) = true;
    commit_received(a_set + ent) =
      (is_in_SentRcvd(CommitReq, ent) implies is_in_SentRcvd(CommitCnf, ent))
      and
      commit_received(a_set);
    rollback_received(empty_assocs) = true;
    rollback_received(a_set + ent) =
      ( is_in_SentRcvd(RollbackInd, ent)
        or
        is_in_SentRcvd(RollbackCnf, ent)
        or
        not(attached(ent))
      )
      and
      rollback_received(a_set);
endtype
(* ---------------------------------------------------------------------------

module last_ready_commit_rollback_received is TPPM_info

    function last_ready_received (S: set_of_assocs, TPPM: tppm_info) : bool is
      is_in_SentRcvd (TPCommit_requ, TPPM) and ready_received (S) endfunc
    function last_commit_received (S: set_of_assocs, TPPM: tppm_info) : bool is
      not (OwesTPDone (TPPM)) and commit_received (S) endfunc
    function last_rollback_received (S: set_of_assocs, TPPM: tppm_info) : bool is
      not (OwesTPDone (TPPM)) and rollback_received (S) endfunc

    function ready_received (S: set_of_assocs) : bool is
        (S match empty_assocs) orelse
        ( (is_in_SentRcvd (AFPrepare_requ, select ENT in S) implies
           is_in_SentRcvd (CReady_indi,select ENT in S))
         and ready_received (select SET in S)) endfunc

    function commit_received (S: set_of_assocs) : bool is
        (S match empty_assocs) orelse
        ((is_in_SentRcvd (CommitReq, select ENT in S) implies
          is_in_SentRcvd (CommitCnf, select ENT in S))
         and commit_received (select SET in S)) endfunc

    function rollback_received (S: set_of_assocs) : bool is
        (S match empty_assocs) orelse
        ((is_in_SentRcvd (RollbackInd, select ENT in S) or
          is_in_SentRcvd (RollbackCnf, select ENT in S) or
          not (attached (select ENT in S))
         ) and rollback_received (select SET in S)) endfunc
endmod
========================================================================== *)

(*
 *      H.5.6   SACF Data Structures
 *)
type SACF_info is ServicePrim, dialogue_channel_identifiers
sorts AMassocPars, sf_info, sacf_state
opns
(*      am_assoc_params - static information about an association which is needed
when selecting an association for a dialogue or channel *)
  AMassocPars (*! constructor *) :
    address, recovery_context_handle_Opt, application_context_name, QOS_ISO8326 -> AMassocPars
  get_addr : AMassocPars -> address
  get_rchO : AMassocPars -> recovery_context_handle_Opt
  get_acn : AMassocPars -> application_context_name
  get_qos : AMassocPars -> QOS_ISO8326

(* sf_info - state information needed by the SACF filter *)
  sf_info (*! constructor *) : dc_id,
    bool, (* contention winner *)
    bool, (* Bid Mandatory *)
    bool, (* is queuing *)
    bool, (* CFear *)
    bool, (* token owed  *)
    bool, (* BidRspSent *)
    bool, (* Send KeepToken *)
    bool, (* CCR token requested  *)
    correlator_Opt, (*   sent  *)
    correlator_Opt, (*  received  *)
    sacf_state -> sf_info
  new_sf_info : bool (* cw *), bool (* bm*)           -> sf_info
  IsWinner,
  IsLoser,
  get_bm,
  IsQing,
  IsCFear,
  IsTokOwed,
  WasBidRspSent,
  SendKeepTok,
  WasTokReq : sf_info -> bool
  Qing,
  TokOwed,
  BidRspSent,
  TokReq, (* Set the appropriate flag to true *)
  NoQ,
  NotTokOwed,
  NotTokReq : sf_info -> sf_info
(* Set flags to false -  BidRspSent reset when change to FREE state *)
  set_CFear, set_skt : bool, sf_info -> sf_info
  set_did : dc_id, sf_info -> sf_info
  change_lbdsent, change_lbdrec : correlator_Opt, sf_info -> sf_info
  inc_lbdsent : sf_info -> sf_info
  get_lbdsent : sf_info -> correlator_Opt
  get_lbdrec : sf_info -> correlator_Opt
  get_did : sf_info -> dc_id

(*sacf_state*)
  FREE (*! constructor *),
  STRAY (*! constructor *),
  BIDDING (*! constructor *),
  BIDCnfRec (*! constructor *),
  BIDIndRec (*! constructor *),
  CL_BeginInd_EX (*! constructor *),
  CL_RollbackCnf_EX (*! constructor *),
  CL_RollbackInd_EX (*! constructor *),
  BUSY (*! constructor *) : -> sacf_state
  sacf_state : status -> sacf_state
  get_state : ServicePrim -> sacf_state
  get_state : sf_info -> sacf_state
  change_state : sacf_state, sf_info -> sf_info
  IsFREE,
  IsSTRAY,
  IsBIDDING,
  IsBIDIndRec,
  IsBIDCnfRec,
  IsCL_BgnInd_EX,
  IsCL_RbkInd_EX,
  IsCL_RbkCnf_EX,
  IsBUSY : sf_info -> bool
  ssid : sacf_state -> Nat
  IsFREE, IsCL_BgnInd_EX, IsCL_RbkInd_EX, IsCL_RbkCnf_EX : sacf_state -> bool
eqns
    forall
      addr : address,
      rchO : recovery_context_handle_Opt,
      acn : application_context_name,
      qos : QOS_ISO8326,
      qosO : quality_of_service_Opt,
      info : sf_info,
      did, did1 : dc_id,
      bl, cw, bm, qing, cfr, to, bs, skt, ctr : bool,
      lbdsent, lbdrec, dcc : correlator_Opt,
      state, state1 : sacf_state,
      sp :
      ServicePrim
(* AMassocPars components *)
  ofsort address get_addr(AMassocPars(addr, rchO, acn, qos)) = addr;
  ofsort recovery_context_handle_Opt
    get_rchO(AMassocPars(addr, rchO, acn, qos)) = rchO;
  ofsort application_context_name
    get_acn(AMassocPars(addr, rchO, acn, qos)) = acn;
  ofsort QOS_ISO8326 get_qos(AMassocPars(addr, rchO, acn, qos)) = qos;
  ofsort bool
    IsFREE(info) = ssid(get_state(info)) eq ssid(FREE);
    IsSTRAY(info) = ssid(get_state(info)) eq ssid(STRAY);
    IsBIDDING(info) = ssid(get_state(info)) eq ssid(BIDDING);
    IsBIDCnfRec(info) = ssid(get_state(info)) eq ssid(BIDCnfRec);
    IsBIDIndRec(info) = ssid(get_state(info)) eq ssid(BIDIndRec);
    IsBUSY(info) = ssid(get_state(info)) eq ssid(BUSY);
    IsCL_RbkInd_EX(info) = ssid(get_state(info)) eq ssid(CL_RollbackInd_EX);
    IsCL_BgnInd_EX(info) = ssid(get_state(info)) eq ssid(CL_BeginInd_EX);
    IsCL_RbkCnf_EX(info) = ssid(get_state(info)) eq ssid(CL_RollbackCnf_EX);
    IsFREE(state) = ssid(state) eq ssid(FREE);
    IsCL_RbkInd_EX(state) = ssid(state) eq ssid(CL_RollbackInd_EX);
    IsCL_BgnInd_EX(state) = ssid(state) eq ssid(CL_BeginInd_EX);
    IsCL_RbkCnf_EX(state) = ssid(state) eq ssid(CL_RollbackCnf_EX);
  ofsort Nat
    ssid(FREE) = 0;
    ssid(STRAY) = succ(ssid(FREE));
    ssid(BIDDING) = succ(ssid(STRAY));
    ssid(BIDCnfRec) = succ(ssid(BIDDING));
    ssid(BIDIndRec) = succ(ssid(BIDCnfRec));
    ssid(BUSY) = succ(ssid(BIDIndRec));
    ssid(CL_RollbackInd_EX) = succ(ssid(BUSY));
    ssid(CL_BeginInd_EX) = succ(ssid(CL_RollbackInd_EX));
    ssid(CL_RollbackCnf_EX) = succ(ssid(CL_BeginInd_EX));
  ofsort sf_info
    new_sf_info(cw, bm) =
      sf_info( did_absent, cw, bm,
 (* winner only must has token initially  if ccr selected *)
        false,
        false,
        false,
        false,
        false,
        false,
        dcc_absent, dcc_absent, FREE)
  ofsort dc_id
    get_did( sf_info(did, cw, bm, qing, cfr, to, bs, skt, ctr, lbdsent, lbdrec, state))
    = did;
  ofsort correlator_Opt
    get_lbdrec( sf_info(did, cw, bm, qing, cfr, to, bs, skt, ctr, lbdsent, lbdrec, state))
    = lbdrec;
    get_lbdsent( sf_info(did, cw, bm, qing, cfr, to, bs, skt, ctr, lbdsent, lbdrec, state))
    = lbdsent;
  ofsort sacf_state
    get_state( sf_info(did, cw, bm, qing, cfr, to, bs, skt, ctr, lbdsent, lbdrec, state))
    = state;
    sacf_state(free) = FREE;
    sacf_state(begin_fear) = FREE;
    sacf_state(rollback_indication_expected) = CL_RollbackInd_EX;
    sacf_state(rollback_confirm_expected) = CL_RollbackCnf_EX;
    sacf_state(begin_indication_expected) = CL_BeginInd_EX;
    get_state(sp) = sacf_state(get_status(sp));
  ofsort bool
    IsWinner( sf_info(did, cw, bm, qing, cfr, to, bs, skt, ctr, lbdsent, lbdrec, state))
    = cw;
    IsLoser( sf_info(did, cw, bm, qing, cfr, to, bs, skt, ctr, lbdsent, lbdrec, state))
    = not(cw);
    get_bm( sf_info(did, cw, bm, qing, cfr, to, bs, skt, ctr, lbdsent, lbdrec, state))
    = bm;
    IsQing( sf_info(did, cw, bm, qing, cfr, to, bs, skt, ctr, lbdsent, lbdrec, state))
    = qing;
    IsCFear( sf_info(did, cw, bm, qing, cfr, to, bs, skt, ctr, lbdsent, lbdrec, state))
    = cfr;
    IsTokOwed( sf_info(did, cw, bm, qing, cfr, to, bs, skt, ctr, lbdsent, lbdrec, state))
    = to;
    WasBidRspSent( sf_info(did, cw, bm, qing, cfr, to, bs, skt, ctr, lbdsent, lbdrec, state))
    = bs;
    SendKeepTok( sf_info(did, cw, bm, qing, cfr, to, bs, skt, ctr, lbdsent, lbdrec, state))
    = skt;
    WasTokReq( sf_info(did, cw, bm, qing, cfr, to, bs, skt, ctr, lbdsent, lbdrec, state))
    = ctr;
  ofsort sf_info
    Qing( sf_info(did, cw, bm, qing, cfr, to, bs, skt, ctr, lbdsent, lbdrec, state))
    = sf_info(did, cw, bm, true, cfr, to, bs, skt, ctr, lbdsent, lbdrec, state);
    TokOwed( sf_info(did, cw, bm, qing, cfr, to, bs, skt, ctr, lbdsent, lbdrec, state))
    = sf_info( did, cw, bm, qing, cfr, true, bs, skt, ctr, lbdsent, lbdrec, state);
    BidRspSent( sf_info(did, cw, bm, qing, cfr, to, bs, skt, ctr, lbdsent, lbdrec, state))
    = sf_info( did, cw, bm, qing, cfr, to, true, skt, ctr, lbdsent, lbdrec, state);
    set_CFear( bl, sf_info(did, cw, bm, qing, cfr, to, bs, skt, ctr, lbdsent, lbdrec, state))
    = sf_info(did, cw, bm, qing, bl, to, bs, bl, ctr, lbdsent, lbdrec, state);
    set_skt( bl, sf_info(did, cw, bm, qing, cfr, to, bs, skt, ctr, lbdsent, lbdrec, state))
    = sf_info(did, cw, bm, qing, cfr, to, bs, bl, ctr, lbdsent, lbdrec, state);
    TokReq( sf_info(did, cw, bm, qing, cfr, to, bs, skt, ctr, lbdsent, lbdrec, state))
    = sf_info( did, cw, bm, qing, cfr, to, bs, skt, true, lbdsent, lbdrec, state);
    NoQ( sf_info(did, cw, bm, qing, cfr, to, bs, skt, ctr, lbdsent, lbdrec, state))
    = sf_info( did, cw, bm, false, cfr, to, bs, skt, ctr, lbdsent, lbdrec, state);
    NotTokOwed( sf_info(did, cw, bm, qing, cfr, to, bs, skt, ctr, lbdsent, lbdrec, state))
    = sf_info( did, cw, bm, qing, cfr, false, bs, skt, ctr, lbdsent, lbdrec, state);
    NotTokReq( sf_info(did, cw, bm, qing, cfr, to, bs, skt, ctr, lbdsent, lbdrec, state))
    = sf_info( did, cw, bm, qing, cfr, to, bs, skt, false, lbdsent, lbdrec, state);
    set_did( did1, sf_info(did, cw, bm, qing, cfr, to, bs, skt, ctr, lbdsent, lbdrec, state))
    = sf_info( did1, cw, bm, qing, cfr, to, bs, skt, ctr, lbdsent, lbdrec, state);
    IsFREE(state1) =>
      change_state( state1, sf_info( did, cw, bm, qing, cfr, to, bs, skt, ctr, lbdsent, lbdrec, state)) =
        sf_info( did, cw, bm, qing, cfr, false, false, false, false, lbdsent, lbdrec, state1);
    not(IsFREE(state1)) =>
      change_state( state1, sf_info( did, cw, bm, qing, cfr, to, bs, skt, ctr, lbdsent, lbdrec, state)) =
        sf_info( did, cw, bm, qing, cfr, to, bs, skt, ctr, lbdsent, lbdrec, state1);
    change_lbdsent( dcc, sf_info(did, cw, bm, qing, cfr, to, bs, skt, ctr, lbdsent, lbdrec, state))
    = sf_info(did, cw, bm, qing, cfr, to, bs, skt, ctr, dcc, lbdrec, state);
    change_lbdrec( dcc, sf_info(did, cw, bm, qing, cfr, to, bs, skt, ctr, lbdsent, lbdrec, state))
    = sf_info(did, cw, bm, qing, cfr, to, bs, skt, ctr, lbdsent, dcc, state);
    inc_lbdsent( sf_info(did, cw, bm, qing, cfr, to, bs, skt, ctr, lbdsent, lbdrec, state))
    = sf_info( did, cw, bm, qing, cfr, to, bs, skt, ctr, inc_dcc(lbdsent), lbdrec, state);
endtype
(* ------------------------------------------------------------------------
module SACF_info is ServicePrim, dialogue_channel_identifiers
/* AMassocPars */
    type AMassocPars is
        AMassocPars (A: address,
                RCHO: recovery_context_handle_Opt,
                ACN: application_context_name,
                QOS QOS_ISO8326)
    endtype
    function get_addr (AMP: AMassocPars) : address is select A in AMP endfunc
    function get_rchO (AMP: AMassocPars) : recovery_context_handle_Opt is select RCHO in AMP endfunc
    function get_acn  (AMP: AMassocPars) : application_context_name is select ACN in AMP endfunc
    function get_qos  (AMP: : AMassocPars) : QOS_ISO8326 is select QOS in AMP endfunc

/*sacf_state*/
    type sacf_state is
        FREE
    |   STRAY
    |   BIDDING
    |   BIDCnfRec
    |   BIDIndRec
    |   CL_BeginInd_EX
    |   CL_RollbackCnf_EX
    |   CL_RollbackInd_EX
    |   BUSY
    endtype

    function sacf_state (S: status) : sacf_state is
        case S in
            free
        |   begin_fear -> FREE
        |   rollback_indication_expected -> CL_RollbackInd_EX
        |   rollback_confirm_expected -> CL_RollbackCnf_EX
        |   begin_indication_expected -> CL_BeginInd_EX
        endcase
    endfunc
    function get_state (SP: ServicePrim) : sacf_state is sacf_state (get_status (SP)) endfunc
/* no translation is needed for function ssid */
    function IsFREE (S: sacf_state) : bool is S match FREE endfunc
    function IsCL_BgnInd_EX (S: sacf_state) : bool is S match CL_BgnInd_EX  endfunc
    function IsCL_RbkInd_EX (S: sacf_state) : bool is S match CL_RbkInd_EX endfunc
    function IsCL_RbkCnf_EX (S: sacf_state) : bool is S match CL_RbkCnf_EX endfunc

/* sf_info */
    type sf_info is
        sf_info (DID: dc_id,
                CW: bool,
                BM: bool,
                QING: bool,
                CFR: bool,
                TO: bool,
                BS: bool,
                SKT: bool,
                CTR: bool,
                LBDSENT: correlator_Opt,
                LBDREC: correlator_Opt,
                STATE: sacf_state)
    endtype

    function new_sf_info (CW: bool, BM: bool) : sf_info is
        sf_info( did_absent, cw, bm, false, false, false, false, false, false,
        dcc_absent, dcc_absent, FREE) endfunc
    function get_state (Info: sf_info) : sacf_state is select STATE in Info endfunc
    function IsFREE (Info: sf_info) : bool is (select STATE in Info) match FREE endfunc
    function IsSTRAY (Info: sf_info) : bool is (select STATE in Info) match STRAY endfunc
    function IsBIDDING (Info: sf_info) : bool is (select STATE in Info) match BIDDING endfunc
    function IsBIDIndRec (Info: sf_info) : bool is (select STATE in Info) match BIDIndRec endfunc
    function IsBIDCnfRec (Info: sf_info) : bool is (select STATE in Info) match BIDCnfRec endfunc
    function IsCL_BgnInd_EX (Info: sf_info) : bool is (select STATE in Info) match CL_BgnInd_EX endfunc
    function IsCL_RbkInd_EX (Info: sf_info) : bool is (select STATE in Info) match CL_RbkInd_EX endfunc
    function IsCL_RbkCnf_EX (Info: sf_info) : bool is (select STATE in Info) match CL_RbkCnf_EX endfunc
    function IsBUSY (Info: sf_info) : bool is (select STATE in Info) match BUSY endfunc

    function get_lbdsent (Info: sf_info) : correlator_Opt is select LBDSENT in Info endfunc
    function get_lbdrec (Info: sf_info) : correlator_Opt is select LBDRES in Info endfunc
    function get_did (Info: sf_info) : dc_id is select DID in Info endfunc
    function IsWinner (Info: sf_info) : bool is select CW in Info endfunc
    function IsLoser (Info: sf_info) : bool is not (select CW in Info) endfunc
    function get_bm (Info: sf_info) : bool is select BM in Info endfunc
    function IsQing (Info: sf_info) : bool is select QING in Info endfunc
    function IsCFear (Info: sf_info) : bool is select CFR in Info endfunc
    function IsTokOwed (Info: sf_info) : bool is select TO in Info endfunc
    function WasBidRspSent (Info: sf_info) : bool is select BS in Info endfunc
    function SendKeepTok (Info: sf_info) : bool is select SKT in Info endfunc
    function WasTokReq (Info: sf_info) : bool is select CTR in Info endfunc

    function Qing (Info: sf_info) : sf_info is S.{QING:=true} endfunc
    function TokOwed (Info: sf_info) : sf_info is S.{TO:=true} endfunc
    function BidRspSent (Info: sf_info) : sf_info is S.{BS:=true} endfunc
    function set_CFear (B: bool, Info: sf_info) : sf_info is S.{CFR:=B} endfunc
    function set_skt (B: bool, Info: sf_info) : sf_info is S.{SKT:=B} endfunc
    function TokReq (Info: sf_info) : sf_info is S.{CTRrue} endfunc
    function NoQ (Info: sf_info) : sf_info is S.{QING:=false} endfunc
    function NotTokOwed (Info: sf_info) : sf_info is S.{TO:=false} endfunc
    function NotTokReq (Info: sf_info) : sf_info is S.{CTR:=false} endfunc
    function set_did (D: dc_id, Info: sf_info) : sf_info is S.{DID:=D} endfunc
    function change_lbdsent (DCC: correlator_Opt, Info: sf_info) : sf_info is S.{LBDSENT:=DCC} endfunc
    function change_lbdrec (DCC: correlator_Opt, Info: sf_info) : sf_info is S.{LBDREC:=DCC} endfunc
    function inc_lbdsent (Info: sf_info) : sf_info is S.{LBDSENT:=inc_dcc (select LBDSENT in Info)} endfunc
    function change_state (S: sacf_state, Info: sf_info) : sf_info is
        if (S match FREE) then Info.{TO:=false, BS:=false, SKT:=false, CTR:=false, STATE:=S}
        else Info.{STATE:=S} endif
    endfunc

endmod
========================================================================= *)

(*
 *      H.5.7   Derived Operations
 *)
type make_mapping_parameter_from_ServicePrimKey is SpecialKeys
opns
  key : mapping -> ServicePrimKey
  make_map : ServicePrim -> mapping
  make_map : ServicePrimKey -> mapping
  make_map : recovery_state -> mapping
  _eq_, _ne_ : mapping, mapping -> bool
  is_map_abortRI,
  is_map_dataRI,
  is_map_commitRI,
  is_map_commitRC,
  is_map_rollbackRI,
  is_map_rollbackRC,
  is_map_recover_doneRC :
    ServicePrim -> bool
eqns
    forall map, map1 : mapping, sp : ServicePrim, spk: ServicePrimKey
  ofsort ServicePrimKey
    key(commitRI) = CCommit_req;
    key(commitRC) = CCommit_rsp;
    key(rollbackRI) = CRollback_req;
    key(rollbackRC) = CRollback_rsp;
    key(recover_doneRC) = CRecover_rsp;
    key(recover_commitRI) = CRecover_req;
    key(recover_readyRI) = CRecover_req;
    key(abortRI) = AAbort_req;
    key(dataRI) = PData_req;
  ofsort mapping
    IsCRecoverInd(sp) => make_map(sp) = make_map(get_rsri(sp));
    not(IsCRecoverInd(sp)) => make_map(sp) = make_map(key(sp));
    spk eq AAbort_ind  => make_map(spk) = abortRI;
    spk eq CCommit_ind => make_map(spk) = commitRI;
    spk eq CCommit_cnf => make_map(spk) = commitRC;
    spk eq CRollback_ind => make_map(spk) = rollbackRI;
    spk eq CRollback_cnf => make_map(spk) = rollbackRC;
    spk eq CRecover_cnf => make_map(spk) = recover_doneRC;
    make_map(commit) = recover_commitRI;
    make_map(ready) = recover_readyRI;
  ofsort bool
    map eq map1 = key(map) eq key(map1);
    map ne map1 = not(map eq map1);
    is_map_abortRI(sp) = get_map(sp) eq abortRI;
    is_map_dataRI(sp) = get_map(sp) eq dataRI;
    is_map_commitRI(sp) = get_map(sp) eq commitRI;
    is_map_commitRC(sp) = get_map(sp) eq commitRC;
    is_map_rollbackRI(sp) = get_map(sp) eq rollbackRI;
    is_map_rollbackRC(sp) = get_map(sp) eq rollbackRC;
    is_map_recover_doneRC(sp) = get_map(sp) eq recover_doneRC;
  ofsort recovery_state
    get_map(sp) = recover_readyRI => get_rsri(sp) = ready;
    get_map(sp) = recover_commitRI => get_rsri(sp) = commit
endtype
(* ------------------------------------------------------------------------
module make_mapping_parameter_from_ServicePrimKey is SpecialKeys

    function key (MAP: mapping) : ServicePrimKey is
        case MAP in
            commitRI -> CCommit_req
        |   commitRC -> CCommit_rsp
        |   rollbackRI -> CRollback_req
        |   rollbackRC -> CRollback_rsp
        |   recover_doneRC -> CRecover_rsp
        |   recover_commitRI -> CRecover_req
        |   recover_readyRI -> CRecover_req
        |   abortRI -> AAbort_req
        |   dataRI -> PData_req
        endcase
    endfunc
    function make_map (SP: ServicePrim) : mapping is
        if IsCRecoverInd (SP) then make_map (get_rsri (SP))
        else make_map (key (SP)) endif
    endfunc
    function make_map (SPK: ServicePrimKey) : mapping is
        case SPK in
            AAbort_ind -> abortRI
        |   CCommit_ind -> commitRI
        |   CCommit_cnf -> commitRC
        |   CRollback_ind -> rollbackRI
        |   CRollback_cnf -> rollbackRC
        |   CRecover_cnf -> recover_doneRC
        endcase
    endfunc
    function make_map (RS: recovery_state) : mapping is
        case RS in
            commit -> recover_commitRI
        |   ready -> recover_readyRI
        endcase
    endfunc
/* synatctical equality will be automatically generated */
    function _ne_ (MAP1: mapping, MAP2: mapping) : bool is not (MAP1 eq MAP2) endfunc

    function is_map_abortRI (SP: ServicePrim) : bool is get_map (SP) eq abortRI endfunc
    function is_map_dataRI (SP: ServicePrim) : bool is get_map (SP) eq dataRI endfunc
    function is_map_commitRI (SP: ServicePrim) : bool is get_map (SP) eq commitRI endfunc
    function is_map_commitRC (SP: ServicePrim) : bool is get_map (SP) eq commitRC endfunc
    function is_map_rollbackRI (SP: ServicePrim) : bool is get_map (SP) eq rollbackRI endfunc
    function is_map_rollbackRC (SP: ServicePrim) : bool is get_map (SP) eq rollbackRC endfunc
    function is_map_recover_doneRC (SP: ServicePrim) : bool is get_map (SP) eq recover_doneRC endfunc

/*  get_map(sp) = recover_readyRI => get_rsri(sp) = ready;
    get_map(sp) = recover_commitRI => get_rsri(sp) = commit *?

endmod
=========================================================================== *)

type make_PDU_from_ServicePrim is PDU, ServicePrim
opns
  make_tp_abort_ri : ServicePrim -> PDU
  make_tp_heuristic_report_ri : ServicePrim -> PDU
  make_AFAbortInd : PDU, mapping -> ServicePrim
  make_AFAbortAndHeuristicReportInd : PDU, PDU, mapping -> ServicePrim
eqns
    forall pdu, pdu1 : PDU, afsp : ServicePrim, map : mapping
  ofsort PDU
```

```
    is_user(afsp) => make_tp_abort_ri(afsp) = tp_abort_ri(user, get_ud(afsp));
    is_provider(afsp) =>
      make_tp_abort_ri(afsp) = tp_abort_ri(provider, get_pad(afsp));
    make_tp_heuristic_report_ri(afsp) = tp_heuristic_report_ri(get_hr(afsp));
  ofsort ServicePrim
    is_user(pdu) =>
      make_AFAbortInd(pdu, map) = AFAbortInd(user, map, get_ud(pdu));
    is_provider(pdu) =>
      make_AFAbortInd(pdu, map) = AFAbortInd(provider, map, get_pad(pdu));
    make_AFAbortAndHeuristicReportInd(pdu1, pdu, map) =
      AFAbortAndHeuristicReportInd(map, get_hr(pdu1), get_ud(pdu));
endtype
(* ---------------------------------------------------------------------------
module make_PDU_from_ServicePrim is PDU, ServicePrim
    function make_tp_abort_ri (SP: ServicePrim) : PDU is
        if is_user (SP) then tp_abort_ri (user, get_ud (SP))
        elseif is_provider (SP) then tp_abort_ri (provider, get_pad (SP)) endif
    endfunc
    function make_tp_heuristic_report_ri (SP: ServicePrim) : PDU is
        tp_heuristic_report_ri (get_hr (SP)) endif
    function make_AFAbortInd (P: PDU, MAP: mapping) : ServicePrim is
        if is_user (P) then AFAbortInd (user, map, get_ud (P))
        elseif is_provider (P) then AFAbortInd (provider, MAP, get_pad (P)) endif
    endfunc

    function make_AFAbortAndHeuristicReportInd (P1: PDU, P2: PDU, MAP: mapping) : ServicePrim is
        AFAbortAndHeuristicReportInd (MAP, get_hr (P1), get_ud (P2)) endfunc
endmod
======================================================================== *)

type ServicePrim_embedded_PDUs is ServicePrim, PDU_Sequences
opns
  get_PDUqueue : ServicePrim -> PDUqueue
  get_pdu, get_pdu2 : ServicePrim -> PDU
  has_PDUqueue,
  has_embedded_pdus,
  has_embedded_pdu,
  has_embedded_BeginDialogueRC,
  has_embedded_Abort,
  has_embedded_HeuristicReport,
  has_embedded_AbortAndHeuristicReport,
  has_embedded_Prepare,
  has_embedded_Recover,
  has_embedded_AssocEstabRI,
  has_embedded_AssocEstabRC,
  has_embedded_TokenGive,
  has_embedded_AARQ : ServicePrim -> bool
  has_embedded : PDUkey, ServicePrim -> bool
  is_Abort_HeuristicReport : PDUqueue -> bool
  _eq_ : PDUqueue, PDUkey -> bool
eqns
    forall sp : ServicePrim, x : PDUkey, q : PDUqueue
  ofsort PDUqueue
    not(IsACSEsp(sp)) => get_PDUqueue(sp) = get_ud(sp);
    IsACSEsp(sp) => get_PDUqueue(sp) = get_aui(sp);
  ofsort PDU
    get_pdu(sp) = head(get_PDUqueue(sp));
    get_pdu2(sp) = head(tail(get_PDUqueue(sp)));
  ofsort bool
    has_PDUqueue(sp) =
      IsAAbortReq(sp) or
      IsAAbortInd(sp) or
      IsAAssocReq(sp) or
      IsAAssocInd(sp) or
      IsAAssocRsp(sp) or
      IsAAssocCnf(sp) or
      IsPWithEmbeddedAPDUReqRsp(sp) or
      IsPWithEmbeddedAPDUIndCnf(sp) or
      IsPDataReq(sp) or
      IsPDataInd(sp) or
      IsPTokenGiveReq(sp) or
      IsPTokenGiveInd(sp) or
      IsCPrepareReq(sp) or
      IsCPrepareInd(sp) or
      IsCCommitReq(sp) or
      IsCCommitInd(sp) or
      IsCCommitRsp(sp) or
      IsCCommitCnf(sp) or
      IsCRollbackReq(sp) or
      IsCRollbackInd(sp) or
      IsCRollbackRsp(sp) or
      IsCRollbackCnf(sp) or
      IsCRecoverReq(sp) or
      IsCRecoverInd(sp) or
      IsCRecoverRsp(sp) or
      IsCRecoverCnf(sp);
    has_embedded_pdus(sp) = has_PDUqueue(sp) and not(IsEmpty(get_PDUqueue(sp)));
    has_embedded_pdu(sp) = has_PDUqueue(sp) and is_PDU(get_PDUqueue(sp));
    has_embedded_AbortAndHeuristicReport(sp) =
      has_PDUqueue(sp) and is_Abort_HeuristicReport(get_PDUqueue(sp));
    has_embedded(x, sp) = has_PDUqueue(sp) and (get_PDUqueue(sp) eq x);
    is_Abort_HeuristicReport(q) = q eq tp_abort_ri_seq;
    q eq x = (head(q) eq x) and IsEmpty(tail(q));
    has_embedded_BeginDialogueRC(sp) =
      has_embedded(tp_begin_dialogue_rc_pdu, sp);
    has_embedded_Abort(sp) = has_embedded(tp_abort_ri_pdu, sp);
    has_embedded_HeuristicReport(sp) =
      has_embedded(tp_heuristic_report_ri_pdu, sp);
    has_embedded_Prepare(sp) = has_embedded(tp_prepare_ri_pdu, sp);
    has_embedded_Recover(sp) = has_embedded(tp_recover_ri_pdu, sp);
    has_embedded_AssocEstabRI(sp) = has_embedded(tp_initialize_ri_pdu, sp);
    has_embedded_AssocEstabRC(sp) = has_embedded(tp_initialize_rc_pdu, sp);
    has_embedded_TokenGive(sp) = has_embedded(tp_token_give_ri_pdu, sp);
    has_embedded_AARQ(sp) = has_embedded(AARQ_pdu, sp)
endtype
(* --------------------------------------------------------------------------
module ServicePrim_embedded_PDUs is ServicePrim, PDU_Sequences
    function get_PDUqueue (SP: ServicePrim) : PDUqueue is
```

```
        if not (IsACSEsp (SP)) then get_ud (SP)
        else get_aui (SP) endif
    endfunc
    function get_pdu (SP: ServicePrim) : PDU is head (get_PDUqueue (SP)) endfunc
    function get_pdu2 (SP: ServicePrim) : PDU is head (tail (get_PDUqueue (SP))) endfunc

    function has_PDUqueue (SP: ServicePrim) : bool is
      IsAAbortReq (SP) or
      IsAAbortInd (SP) or
      IsAAssocReq (SP) or
      IsAAssocInd (SP) or
      IsAAssocRsp (SP) or
      IsAAssocCnf (SP) or
      IsPWithEmbeddedAPDUReqRsp (SP) or
      IsPWithEmbeddedAPDUIndCnf (SP) or
      IsPDataReq (SP) or
      IsPDataInd (SP) or
      IsPTokenGiveReq (SP) or
      IsPTokenGiveInd (SP) or
      IsCPrepareReq (SP) or
      IsCPrepareInd (SP) or
      IsCCommitReq (SP) or
      IsCCommitInd (SP) or
      IsCCommitRsp (SP) or
      IsCCommitCnf (SP) or
      IsCRollbackReq (SP) or
      IsCRollbackInd (SP) or
      IsCRollbackRsp (SP) or
      IsCRollbackCnf (SP) or
      IsCRecoverReq (SP) or
      IsCRecoverInd (SP) or
      IsCRecoverRsp (SP) or
      IsCRecoverCnf (SP) endfunc
    function has_embedded_pdus (SP: ServicePrim) : bool is
        has_PDUqueue (SP) and not (IsEmpty (get_PDUqueue (SP))) endfunc
    function  has_embedded_pdu (SP: ServicePrim) : bool is
        has_PDUqueue (SP) and is_PDU (get_PDUqueue (SP)) endfunc
    function has_embedded_AbortAndHeuristicReport (SP: ServicePrim) : bool is
      has_PDUqueue (SP) and is_Abort_HeuristicReport (get_PDUqueue (SP)) endfunc
    function has_embedded (K: PDUkey, SP: ServicePrim) : bool is
        has_PDUqueue (SP) and (get_PDUqueue (SP) eq K) endfunc
    function  _eq_ (Q: PDUqueue, K: PDUkey) : bool is
        (head (Q) eq K) and IsEmpty (tail (Q)) endfunc
    function is_Abort_HeuristicReport (Q: PDUqueue) : bool is
        Q eq tp_abort_ri_seq endfunc
    function has_embedded_BeginDialogueRC (SP: ServicePrim) : bool is
        has_embedded (tp_begin_dialogue_rc_pdu, SP) endfunc
    function has_embedded_Abort (SP: ServicePrim) : bool is
        has_embedded (tp_abort_ri_pdu, SP) endfunc
    function has_embedded_HeuristicReport (SP: ServicePrim) : bool is
        has_embedded (tp_heuristic_report_ri_pdu, SP) endfunc
    function has_embedded_Prepare (SP: ServicePrim) : bool is
        has_embedded (tp_prepare_ri_pdu, SP) endfunc
    function has_embedded_Recover (SP: ServicePrim) : bool is
```

```
        has_embedded (tp_recover_ri_pdu, SP) endfunc
    function has_embedded_AssocEstabRI (SP: ServicePrim) : bool is
        has_embedded (tp_initialize_ri_pdu, SP) endfunc
    function has_embedded_AssocEstabRC (SP: ServicePrim) : bool is
        has_embedded (tp_initialize_rc_pdu, SP) endfunc
    function has_embedded_TokenGive (SP: ServicePrim) : bool is
        has_embedded(tp_token_give_ri_pdu, SP) endfunc
    function has_embedded_AARQ (SP: ServicePrim) : bool is
        has_embedded(AARQ_pdu, SP) endfunc
endmod
============================================================================ *)

type concatentation_sequences is PDUkey_Lists, ServicePrim_embedded_PDUs
opns
  allowed_concat, allowed_tail, allowed_CCR_concat : PDUqueue -> bool
  can_not_concat,
  begin_concat,
  end_concat,
  may_be_concat,
  can_begin,
  can_end,
  can_concat : PDU -> bool
  end_concat : PDU, confirmation_urgency_Opt -> bool
eqns
    forall pdu : PDU, q : PDUqueue, cnfuO : confirmation_urgency_Opt
  ofsort bool
    can_not_concat(pdu) = pdu IsIn
      (tp_bid_ri_pdu . tp_bid_rc_pdu . tp_prepare_ri_pdu . tp_recover_ri_pdu .
        c_rollback_ri_pdu . c_rollback_rc_pdu)
      or istp_begin_dialogue_rc_rejected(pdu);
    begin_concat(pdu) = istp_begin_dialogue_ri(pdu) or istp_begin_dialogue_rc_accepted(pdu);
    end_concat(pdu) = pdu IsIn
      ( tp_end_dialogue_ri_pdu . tp_end_dialogue_rc_pdu . tp_abort_ri_pdu .
        tp_grant_control_ri_pdu . tp_request_control_ri_pdu .
        tp_handshake_ri_pdu . tp_handshake_and_grant_control_ri_pdu);
    may_be_concat(pdu) = pdu IsIn
      ( tp_u_error_ri_pdu . tp_u_error_rc_pdu . tp_token_give_ri_pdu . tp_defer_ri_pdu
        . tp_heuristic_report_ri_pdu . tp_initialize_ri_pdu . tp_initialize_rc_pdu)
      or IsUPDU(pdu)
      or (IsCCRpdu(pdu) and not(can_not_concat(pdu)));
    can_begin(pdu) = begin_concat(pdu) or may_be_concat(pdu);
    can_end(pdu) = may_be_concat(pdu) or end_concat(pdu);
    can_concat(pdu) = begin_concat(pdu) or may_be_concat(pdu) or end_concat(pdu);
    not(is_handshake_rc(pdu)) => end_concat(pdu, cnfuO) = end_concat(pdu);
    is_handshake_rc(pdu) => end_concat(pdu, Opt(urgent)) = true;
    is_handshake_rc(pdu) => end_concat(pdu, cnfu_absent) = true;
    is_handshake_rc(pdu) => end_concat(pdu, Opt(normal)) = false;
    not(IsEmpty(q)) =>
      allowed_concat(pdu . q) = can_begin(pdu) and allowed_tail(q) and allowed_CCR_concat(pdu . q);
(* allowed_CCR_concat not defined by eqautions *)
    allowed_tail(pdu . emptyPDU) = can_end(pdu);
    not(IsEmpty(q)) =>
      allowed_tail(pdu . q) = may_be_concat(pdu) and allowed_tail(q)
endtype
```

```
(* -------------------------------------------------------------------------
module concatentation_sequences is PDUkey_Lists, ServicePrim_embedded_PDUs

    function can_not_concat (P: PDU) : bool is
        P IsIn (tp_bid_ri_pdu . tp_bid_rc_pdu . tp_prepare_ri_pdu
                . tp_recover_ri_pdu . c_rollback_ri_pdu . c_rollback_rc_pdu)
        or istp_begin_dialogue_rc_rejected (P) endfunc
    function begin_concat (P: PDU) : bool is
        istp_begin_dialogue_ri (P) or istp_begin_dialogue_rc_accepted (P) endfunc
    function end_concat (P: PDU) : bool is
        pdu IsIn
        (tp_end_dialogue_ri_pdu . tp_end_dialogue_rc_pdu . tp_abort_ri_pdu
        . tp_grant_control_ri_pdu . tp_request_control_ri_pdu . tp_handshake_ri_pdu
        . tp_handshake_and_grant_control_ri_pdu) endfunc
    function may_be_concat (P: PDU) : bool is
        pdu IsIn
        (tp_u_error_ri_pdu . tp_u_error_rc_pdu . tp_token_give_ri_pdu . tp_defer_ri_pdu
        . tp_heuristic_report_ri_pdu . tp_initialize_ri_pdu . tp_initialize_rc_pdu)
      or IsUPDU (P)
      or (IsCCRpdu (P) and not(can_not_concat (P))) endfunc
    function can_begin (P: PDU) : bool is
        begin_concat (P) or may_be_concat (P) endfunc
    function can_end (P: PDU) : bool is
        may_be_concat (P) or end_concat (P) endfunc
    function can_concat (P: PDU) : bool is
        begin_concat (P) or may_be_concat (P) or end_concat (P) endfunc
    function end_concat (P: PDU, CNFUO: confirmation_urgency_Opt) : bool is
        if not (is_handshake_rc (P)) then end_concat (P)
        else CNFUO match cnfu_absent endif
    not(IsEmpty(q)) =>
    function allowed_concat (Q: PDUqueue) : bool is
        let (P:=P: PDU) . (PQ:=PQ: PDUqueue) = Q in
        not (IsEmpty (PQ)) andthen
        can_begin (P) and allowed_tail (PQ) and allowed_CCR_concat (Q) endfunc
    function allowed_tail (Q: PDUqueue) : bool is
        let (P:=P: PDU) . (PQ:=PQ: PDUqueue) = Q in
        if PQ match emptyPDU then can_end (P)
        else may_be_concat (P) and allowed_tail (PQ) endfunc
endmod
========================================================================== *)

type mapping_to_presentation_ServicePrim is make_PDU_from_ServicePrim
opns
  all_TPpdus : PDUqueue -> bool
  presentation_embedding : PDUqueue -> ServicePrim
  presentation_embedding : PDU -> ServicePrim
eqns
    forall pdu : PDU, q : PDUqueue
  ofsort bool
    all_TPpdus(emptyPDU) = true;
    all_TPpdus(pdu . q) = IsTPpdu(pdu) and all_TPpdus(q);
(*  ofsort ServicePrimKey
    not(all_TPpdus(q)) =>
      key(presentation_embedding(q)) = PWithEmbeddedAPDU_ReqRsp; *)
  ofsort ServicePrim
    presentation_embedding(pdu) = presentation_embedding(pdu . emptyPDU);
    Istp_token_give_ri(pdu) =>
      presentation_embedding(pdu . emptyPDU) = PTokenGiveReq(pdu);
    not(Istp_token_give_ri(head(q))) and all_TPpdus(q) =>
      presentation_embedding(q) = PDataReq(q);
endtype
(* ------------------------------------------------------------------------
module mapping_to_presentation_ServicePrim is make_PDU_from_ServicePrim
    function all_TPpdus (Q: PDUqueue) : bool is
        (Q match emptyPDU) orelse
        (IsTPpdu (select PDU in Q) and all_TPpdus (select PQ in Q)) endfunc
/*  not(all_TPpdus(q)) =>
      key(presentation_embedding(q)) = PWithEmbeddedAPDU_ReqRsp; */
    function presentation_embedding (P: PDU) : ServicePrim is
        presentation_embedding (P . emptyPDU) endfunc
    function presentation_embedding (Q: PDUqueue) : ServicePrim is
        case Q in
            (PDU:=P: PDU) . emptyPDU when Istp_token_give_ri (P) -> PTokenGiveReq (P)
        |   any PDUqueue when not (Istp_token_give_ri (head (Q))) and all_TPpdus (PQ) -> PDataReq (Q)
        endcase
    endfunc
endmod
========================================================================= *)

(*
 *      H.5.8   Lists and Queues
 *)
type ServicePrim_Lists is ServicePrim
sorts ServicePrimList
opns
  nilSP (*! constructor *) : -> ServicePrimList
  _+_ (*! constructor *) : ServicePrimList, ServicePrim -> ServicePrimList
  IsEmpty : ServicePrimList -> bool
  _isin_ : LookUpKey, ServicePrimList -> bool
  _-_ : ServicePrimList, LookUpKey -> ServicePrimList
  remove : LookUpKeyList, ServicePrimList -> ServicePrimList
  get : LookUpKey, ServicePrimList -> ServicePrim
  last : ServicePrimList -> ServicePrim
eqns
    forall
      sp : ServicePrim, x : LookUpKey, l : ServicePrimList, ll : LookUpKeyList
  ofsort bool
    IsEmpty(nilSP) = true;
    IsEmpty(l + sp) = false;
    x IsIn nilSP = false;
    x IsIn (l + sp) = (sp eq x) or (x IsIn l);
  ofsort ServicePrimList
    nilSP - x = nilSP;
    sp eq x => (l + sp) - x = l;
    not(sp eq x) => (l + sp) - x = (l - x) + sp;
    remove(ll, nilSP) = nilSP;
    sp IsIn ll => remove(ll, l + sp) = remove(ll, l);
    not(sp IsIn ll) => remove(ll, l + sp) = remove(ll, l) + sp;
  ofsort ServicePrim
    get(x, nilSP) = absentSP;
    sp eq x => get(x, l + sp) = sp;
    not(sp eq x) => get(x, l + sp) = get(x, l);
    last(l + sp) = sp;
endtype

type ServicePrim_Queues is ServicePrim
sorts ServicePrimQueue
opns
  emptySP (*! constructor *) : -> ServicePrimQueue
  _._ (*! constructor *) : ServicePrim, ServicePrimQueue -> ServicePrimQueue
  _+_ : ServicePrimQueue, ServicePrim -> ServicePrimQueue
  IsEmpty : ServicePrimQueue -> bool
  head : ServicePrimQueue -> ServicePrim
  tail : ServicePrimQueue -> ServicePrimQueue
eqns
    forall sp, sp1 : ServicePrim, q : ServicePrimQueue
  ofsort ServicePrim head(sp . q) = sp;
  ofsort ServicePrimQueue
    tail(sp . q) = q;
    emptySP + sp = sp . emptySP;
    (sp . q) + sp1 = sp . (q + sp1);
  ofsort bool IsEmpty(emptySP) = true; IsEmpty(sp . q) = false;
endtype

type PDUkey_Lists is PDU
sorts PDUlist
opns
  nilPDU (*! constructor *) : -> PDUlist
  _._ (*! constructor *) : PDUlist, PDUkey -> PDUlist
  _._ : PDUkey, PDUkey -> PDUlist
  _isin_ : PDU, PDUlist -> bool
eqns
    forall pdu : PDU, x, x1 : PDUkey, l : PDUlist
  ofsort PDUlist x . x1 = nilPDU . x . x1
  ofsort bool
    pdu IsIn nilPDU = false; pdu IsIn (l . x) = (pdu eq x) or (pdu IsIn l)
endtype

type PDU_Queues is BasicPDU
sorts PDUqueue
opns
  emptyPDU (*! constructor *) : -> PDUqueue
  _._ (*! constructor *) : PDU, PDUqueue -> PDUqueue
  queue : PDU -> PDUqueue
  _+_ : PDUqueue, PDU -> PDUqueue
  _+_ : PDU, PDU -> PDUqueue
  IsEmpty, is_PDU : PDUqueue -> bool
  head : PDUqueue -> PDU
  tail : PDUqueue -> PDUqueue
eqns
    forall pdu, pdu1 : PDU, q : PDUqueue
  ofsort PDU head(pdu . q) = pdu;
  ofsort PDUqueue
    tail(pdu . q) = q;
    queue(pdu) = pdu . emptyPDU;
    emptyPDU + pdu = queue(pdu);
    (pdu . q) + pdu1 = pdu . (q + pdu1);
    pdu + pdu1 = pdu . queue(pdu1);
  ofsort bool
    IsEmpty(emptyPDU) = true;
    IsEmpty(pdu . q) = false;
    is_PDU(q) = not(IsEmpty(q)) and IsEmpty(tail(q));
endtype

type PDU_Sequences is PDU
sorts PDUseq
opns
  empty_seq (*! constructor *) : -> PDUseq
  _+_ (*! constructor *) : PDUkey, PDUseq -> PDUseq
  _eq_ : PDUqueue, PDUseq -> bool
  tp_abort_ri_seq : -> PDUseq
eqns
    forall q : PDUqueue, x : PDUkey, seq : PDUseq
  ofsort bool
    q eq empty_seq = IsEmpty(q);
    q eq (x + seq) = not(IsEmpty(q)) and (head(q) eq x) and (tail(q) eq seq)
  ofsort PDUseq
    tp_abort_ri_seq =
      tp_abort_ri_pdu + (tp_heuristic_report_ri_pdu + empty_seq)
endtype

(* --------------------------------------------------------------------------
module ServicePrim_Lists is ServicePrim
    type  ServicePrimList is
        nilSP
    |   _+_ (SPL: ServicePrimList, SP: ServicePrim)
    endtype
  remove : LookUpKeyList, ServicePrimList -> ServicePrimList
  get : LookUpKey, ServicePrimList -> ServicePrim
  last : ServicePrimList -> ServicePrim

    function IsEmpty (L: ServicePrimList) : bool is L match nilSP endfunc
    function _IsIn_ (K: LookUpKey, L: ServicePrimList) ; bool is
        case L in
            nilSP -> false
        |   (SPL:=L1: ServicePrimList) + (SP:=SP: ServicePrim) ->
                (SP eq K) or (K IsIn L1)
        endcase
    endfunc
    function _-_ (L: ServicePrimList, K: LookUpKey) : ServicePrimList is
        case L in
            nilSP -> nilSP
        |   (SPL:=L1: ServicePrimList) + (SP:=SP: ServicePrim) ->
                if SP eq K then L1 else (L1 - K) + SP endif
        endcase
    endfunc
    function remove (LL: LookUpKeyList, L: ServicePrimList) : ServicePrimList is
        case L in
            nilSP -> nilSP
        |   (SPL:=L1: ServicePrimList) + (SP:=SP: ServicePrim) ->
                if SP IsIn LL then remove (LL, L1)
                else remove(LL, L1) + SP endif
        endcase
    endfunc

    function get (K: LookUpKey, L: ServicePrimList) : ServicePrim is
        case L in
             nilSP -> absentSP
        |    (SPL:=L1: ServicePrimList) + (SP:=SP: ServicePrim) ->
                if SP eq K then SP else get (K, L1) endif
        endcase
    endfunc
    function last (L: ServicePrimList) : ServicePrim is select SP in L endfunc
endmod

module ServicePrim_Queues is ServicePrim
    type  ServicePrimQueue is
        emptySP
    |   _._ (SP: ServicePrim, SPQ: ServicePrimQueue)
    endtype

    function _+_ (Q: ServicePrimQueue, SP1: ServicePrim) : ServicePrimQueue is
        case Q in
            emptySP -> SP1.emptySP
        |   (SP:=SP: ServicePrim).(SPQ:=SPQ: ServicePrimQueue) -> SP.(SPQ+SP1)
        endcase
    endfunc
    fuction IsEmpty (Q: ServicePrimQueue) : bool is Q match emptySP endfunc
    functon head (Q: ServicePrimQueue) : ServicePrim is select SP in Q endfunc
    functon tail (Q: ServicePrimQueue) : ServicePrimQueue is select SPQ in Q endfunc
endmod

module PDUkey_Lists is PDU
    type  PDUlist is
        nilPDU
    | _._ (PL: PDUlist, K: PDUkey)
    endtype
    function _._ (K1: PDUkey, k2: PDUkey) : PDUlist is nilPDU . K1 . K2 endfunc
    function _isin_ (P: PDU, L: PDUlist) : bool is
        case L in
            nilPDU -> false
        |   (PL:=PL: PDUlist, K:=K: PDUkey) -> (P eq K) or (P IsIn PL)
        endcase
    endfunc
endmod

module PDU_Queues is BasicPDU
    type  PDUqueue is
        emptyPDU
    |   _._ (P: PDU, PQ: PDUqueue)
    endtype
    function queue (P: PDU) : PDUqueue is P.emptyPDU endfunc
    function _+_ (Q: PDUqueue, P1: PDU) : PDUqueue is
        case Q in
            emptyPDU ->  P1.emptyPDU
        |   (P:=P: PDU).(PQ:=PQ: PDUqueue) -> P.(PQ+P1)
        endcase
    endfunc
    function _+_ (P1: PDU, P2: PDU) : PDUqueue is P1.queue(P2) endfunc
    function IsEmpty (Q: PDUqueue) : bool is Q match emptyPDU endfunc
    function is_PDU (Q: PDUqueue) : bool is
        not (Q match emptyPDU) and ((select PQ in Q) match emptyPDU) endfunc
    function head (Q: PDUqueue) : PDU is select PQ in Q endfunc
    function tail (Q: PDUqueue) : PDUqueue is select P in Q endfunc
endmod

module PDU_Sequences is PDU
    type  PDUseq is
        empty_seq
    |   _+_ (K: PDUkey, PS: PDUseq)
    endfunc
    function _eq_ (Q: PDUqueue, S: PDUseq) : bool is
        case S in
            empty_seq -> Q match emptyPDU
        |   (K:=K: PDUkey) + (PS:=PS: PDUseq) -> not (Q match emptyPDU)
                and (head (Q) eq K) and (tail (Q) eq PS)
        endcase
    endfunc
    function tp_abort_ri_seq : PDUseq is
         tp_abort_ri_pdu + (tp_heuristic_report_ri_pdu + empty_seq) endfunc
endmod
=========================================================================== *)

(*
 *      H.5.9   Keys
 *)
type CoordinationKeys is NaturalNumber
sorts coord_key, flush_key, log_key, other_key, SACF_key, input_output
opns
  lock (*! constructor *),
  sao_lock (*! constructor *),
  sao_unlock (*! constructor *),
  sacf_int_lock (*! constructor *),
  tppm_crash (*! constructor *),
  close_psap (*! constructor *),
  open_psap (*! constructor *),
  sao_killed (*! constructor *),
  UASE_abort (*! constructor *),
  UASE_PTokReq (*! constructor *),
  UASE_PTokInd (*! constructor *),
  SACF_PE (*! constructor *),
  InternalError (*! constructor *),
  address_not_recognised (*! constructor *),
  Token (*! constructor *) : -> coord_key
  REM (*! constructor *),
  pass_to_UASE (*! constructor *) : -> SACF_key
  service_PE (*! constructor *),
  serviceUser_error (*! constructor *),
  suppress_indication (*! constructor *) : -> other_key
  write_log_record (*! constructor *),
  read_log_record (*! constructor *),
  log_record_absent (*! constructor *),
  forget_transaction (*! constructor *),
  write_log_damage (*! constructor *),
  read_log_damage (*! constructor *),
  log_damage_absent (*! constructor *),
  remove_log_damage (*! constructor *),
  write_log_heuristic (*! constructor *),
  read_log_heuristic (*! constructor *),
  log_heuristic_absent (*! constructor *),
  remove_log_heuristic (*! constructor *),
  branch_list  (*! constructor *): -> log_key
  queue (*! constructor *),
  discard (*! constructor *),
  flush (*! constructor *),
  flush_to_CBegin (*! constructor *) : -> flush_key
  IsFlush, IsDiscard, IsToCBegin : flush_key -> bool
  nat : flush_key -> nat
  up (*! constructor *), dn (*! constructor *) : -> input_output
eqns
    forall k : flush_key
  ofsort nat
    nat(queue) = 0; nat(discard) = 1; nat(flush) = 2; nat(flush_to_Cbegin) = 3;
  ofsort bool
    IsDiscard(k) = nat(k) eq 1;
    IsFlush(k) = nat(k) eq 2;
    IsToCBegin(k) = nat(k) eq 3;
endtype
(* ------------------------------------------------------------------------
module CoordinationKeys is NaturalNumber
sorts coord_key, flush_key, log_key, other_key, SACF_key, input_output
/* coord_key */
    type coord_key is
        lock
    |   sao_lock
    |   sao_unlock
    |   sacf_int_lock
    |   tppm_crash
    |   close_psap
    |   open_psap
    |   sao_killed
    |   UASE_abort
    |   UASE_PTokReq
    |   UASE_PTokInd
    |   SACF_PE
    |   InternalError
    |   address_not_recognised
    |   Token
    endtype
/* SACF_key */
    type  SACF_key is
        REM
    |   pass_to_UASE
    endtype
/* other_key */
    type other_key is
        service_PE
    |   serviceUser_error
    |   suppress_indication
    endtype
/* log_key */
    type log_key is
        write_log_record
    |   read_log_record
    |   log_record_absent
    |   forget_transaction
    |   write_log_damage
    |   read_log_damage
    |   log_damage_absent
    |   remove_log_damage
    |   write_log_heuristic
    |   read_log_heuristic
    |   log_heuristic_absent
    |   remove_log_heuristic
    |   branch_list
    endtype
/* flush_key */
    type flush_key is
        queue
    |   discard
    |   flush
    |   flush_to_CBegin
    endtype

    function IsFlush (FK: flush_key) : bool is FK match flush endfunc
    function IsDiscard (FK: flush_key) : bool is FK match discard endfunc
    function IsToCBegin (FK: flush_key) : bool is FK match flush_to_CBegin endfunc
/* no traslation is needed for function nat */
/*  input_output */
    type input_output is
        up
    |   dn
    endtype
endmod
=========================================================================== *)

(*
 *      H.5.10  Service Constraints Data Structure
 *)
type service_info is ServicePrim, dialogue_channel_identifiers
sorts service_info, assignment, range
opns
```

```
  info (*! constructor *) :
    dc_id, tp_begin_dialogue_confirmation, bit_string, dc_id -> service_info
  get_did, get_next_did : service_info -> dc_id
  get_bdc : service_info -> tp_begin_dialogue_confirmation
  _._ : service_info, nat -> bool
  inc_next_did : service_info -> service_info
  _.._ (*! constructor *) : nat, nat -> range
  _.=_ (*! constructor *),
  _&=_ (*! constructor *),
  _+=_ (*! constructor *) : nat, bool -> assignment
  _.=_ (*! constructor *),
   _&=_ (*! constructor *) : range, bool -> assignment
  _.=_ (*! constructor *) : range, service_info -> assignment  (* copy  *)
  _._ : service_info, assignment -> service_info
  polarized, (* functional unit  *)
  handshake, (*  functional unit  *)
  unchained, (*  functional unit  *)
  failed, (* dialogue terminated *)
  cntrl, (*  has control   *)
  cntrl_trans, (*  had control at start of transaction   *)
  cl_cmt, (*  has coordination level of commitment   *)
  ReqRsp, (*  any requests or responses issued   *)
  IndCnf, (*  any indications or confirmations issued   *)
  DEReq, (*  dialogue establishment request outstanding   *)
  DEInd, (* dialogue establishment indication outstanding   *)
  HSReq, (*  TPHandshakeReq outstanding   *)
  HSInd, (*  TPHandshakeInd outstanding   *)
  HSGCReq, (*  TPHandshakeAndGrantControlReq outstanding   *)
  HSGCInd, (*  TPHandshakeAndGrantControlInd outstanding   *)
  UEReq, (*  user error request outstanding   *)
  UEInd, (*  user error indication outstanding   *)
  DTReq, (*  dialogue termination request outstanding   *)
  DTInd, (* dialogue termination indication outstanding   *)
  Prp_sp, (*  TPPrepare Req/Ind issued   *)
  data_perm, (* data_permitted parameter on TPPrepare  *)
  RdyInd, (*  TPReadyInd issued   *)
  DeferED, (* TPDeferredEndDialogue Req/Ind issued   *)
  DeferGC, (*  TPDeferredGrantControl Req/Ind issued   *)
  HRInd, (* TPHeuristicReportInd issued   *)
  one_dial_cl_cmt, (*  at least one dialogue had coordination level of commitment *)
  CmtReq, (*  TPCommitReq issued   *)
  CmtInd, (*  TPCommitInd issued  *)
  DoneReq, (*  TPDoneReq issued   *)
  Rbk_sp, (*  TPRollback Req/Ind issued   *)
  Rbk_init, (*  any rollback initiating service primitive issued   *)
  AbRej,
(*
    TP[U/P]AbortInd or TPBeginDailogueCnf( rejected) issued and
    subsequent TPDoneReq not yet issued
 *)
  Prepared
(*
    root of transaction tree _or_ TPPrepareInd issued on the
    dialogue with the superior next dialogue
 *)
  : -> nat
  ReqRsp_IndCnf : ServicePrim -> nat
  has_polarized,
  has_shared,
  has_handshake,
  has_commit,
  has_chained,
  has_unchained,
  has_recovery,
  IsMultiReq,
  IsMultiInd,
  IsMulti,
  dial_coord_commitment : ServicePrim -> bool
  init_info : ServicePrim -> service_info
  first_coord_commitment : bool (* superior on dialogue *), service_info -> service_info
  info_sub, info_sup : dc_id, ServicePrim, dc_id -> service_info
  new_info, update, special_update : ServicePrim, service_info -> service_info
eqns
    forall
      did, next_did : dc_id,
      bdc : tp_begin_dialogue_confirmation,
      s, s_copy : service_info,
      n, m : nat,
      b, superior : bool,
      bs : bit_string,
      sp : ServicePrim
  ofsort nat
    polarized = 0;
    handshake = succ(polarized);
    unchained = succ(handshake);
    failed = succ(unchained);
    cntrl = succ(failed);
    cntrl_trans = succ(cntrl);
    cl_cmt = succ(cntrl_trans);
    ReqRsp = succ(cl_cmt);
    IndCnf = succ(ReqRsp);
    DEReq = succ(IndCnf);
    DEInd = succ(DEReq);
    HSReq = succ(DEInd);
    HSInd = succ(HSReq);
    HSGCReq = succ(HSInd);
    HSGCInd = succ(HSGCReq);
    UEReq = succ(HSGCInd);
    UEInd = succ(UEReq);
    DTReq = succ(UEInd);
    DTInd = succ(DTReq);
    Prp_sp = succ(DTInd);
    data_perm = succ(Prp_sp);
    RdyInd = succ(data_perm);
    DeferED = succ(RdyInd);
    DeferGC = succ(DeferED);
    HRInd = succ(DeferGC);
    one_dial_cl_cmt = succ(HRInd);
    CmtReq = succ(one_dial_cl_cmt);
    CmtInd = succ(CmtReq);
    DoneReq = succ(CmtInd);
    Rbk_sp = succ(DoneReq);
    Rbk_init = succ(Rbk_sp);
    AbRej = succ(Rbk_init);
    Prepared = succ(AbRej);
    IsReqRsp(sp) => ReqRsp_IndCnf(sp) = ReqRsp;
    IsIndCnf(sp) => ReqRsp_IndCnf(sp) = IndCnf
  ofsort dc_id
    get_did(info(did, bdc, bs, next_did)) = did;
    get_next_did(info(did, bdc, bs, next_did)) = next_did;
  ofsort tp_begin_dialogue_confirmation
    get_bdc(info(did, bdc, bs, next_did)) = bdc;
  ofsort bool
    has_polarized(sp) = polarized_selected(get_sfu(sp));
    has_shared(sp) = shared_selected(get_sfu(sp));
    has_handshake(sp) = handshake_selected(get_sfu(sp));
    has_commit(sp) = commit_selected(get_sfu(sp));
    has_chained(sp) = chained_selected(get_sfu(sp));
    has_unchained(sp) = unchained_selected(get_sfu(sp));
    has_recovery(sp) = recovery_selected(get_sfu(sp));
    dial_coord_commitment(sp) =
      has_chained(sp) or (has_unchained(sp) and get_bool(get_bt0(sp)));
    IsMultiReq(sp) =
      IsTPCommitReq(sp) or IsTPRollbackReq(sp) or IsTPDoneReq(sp);
    IsMultiInd(sp) =
      IsTPCommitInd(sp)
      or
      IsTPCommitCompleteInd(sp)
      or
      IsTPRollbackInd(sp)
      or
      IsTPRollbackCompleteInd(sp);
    IsMulti(sp) = IsMultiReq(sp) or IsMultiInd(sp);
    info(did, bdc, bs, next_did) . n = get_bit(n, bs);
  ofsort service_info
    inc_next_did(info(did, bdc, bs, next_did)) =
      info(did, bdc, bs, inc(next_did));
    info(did, bdc, bs, next_did) . (n .= b) =
      info(did, bdc, set_bit(n, b, bs), next_did);
    s . (n &= b) = s . (n .= (s . n and b));
    s . (n += b) = s . (n .= (s . n or b));
    n eq m => s . (n .. m .= b) = s . (n .= b);
    n lt m => s . (n .. m .= b) = s . (n .= b) . (succ(n) .. m .= b);
    s . (n .. m &= true) = s;
    s . (n .. m &= false) = s . (n .. m .= false);
    n eq m => s . (n .. m .= s_copy) = s . (n .= (s_copy . n));
    n lt m =>
      s . (n .. m .= s_copy) = s . (n .= (s_copy . n)) . (succ(n) .. m .= s_copy);
    IsTPBeginDialogueReq(sp) and dial_coord_commitment(sp) =>
      init_info(sp) =
        first_coord_commitment( true, info_sub(first_sub_dial, sp, inc(first_sub_dial)));
    IsTPBeginDialogueReq(sp) and not(dial_coord_commitment(sp)) =>
```

```
      init_info(sp) = info_sub(first_sub_dial, sp, inc(first_sub_dial));
    info_sub(did, sp, next_did) =
      info( did, get_bdc(sp),
        has_polarized(sp) + (has_handshake(sp) + string(has_unchained(sp))),
        next_did) .
      (cntrl .= true) .
      (cntrl_trans .= true) .
      (cl_cmt .= dial_coord_commitment(sp)) .
      (ReqRsp .= true) .
      (DEReq .= is_always(get_bdc(sp)));
    IsTPBeginDialogueInd(sp) and dial_coord_commitment(sp) =>
      init_info(sp) =
        first_coord_commitment(false, info_sup(sup_dial, sp, inc(sup_dial)));
    IsTPBeginDialogueInd(sp) and not(dial_coord_commitment(sp)) =>
      init_info(sp) = info_sup(sup_dial, sp, inc(sup_dial));
    info_sup(did, sp, next_did) =
      info( did, get_bdc(sp), has_polarized(sp) + (has_handshake(sp) + string(has_unchained(sp))),
        next_did) .
      (cl_cmt .= dial_coord_commitment(sp)) .
      (IndCnf .= true) .
      (DEInd .= is_always(get_bdc(sp)));
    IsTPBeginDialogueReq(sp) =>
      new_info(sp, s) = info_sub(get_next_did(s), sp, did_absent).(one_dial_cl_cmt .. prepared .= s);
    IsTPDataReq(sp) or
    IsTPDataInd(sp) or
    IsTPRequestControlReq(sp) or
    IsTPRequestControlInd(sp) => update(sp, s) = s . (ReqRsp_IndCnf(sp) .= true);
    IsTPBeginDialogueReq(sp) =>
(*
effects on the first dialogue of sending TPBeginDialogueReq to start a new
dialogue
 *)
      update(sp, s) = inc_next_did(s) . (ReqRsp .= true);
    IsTPBeginDialogueRsp_accepted(sp) =>
      update(sp, s) = s . (ReqRsp .= true) . (DEInd .= false);
    IsTPBeginDialogueRsp_rejected(sp) or (s . DTInd and IsTPPAbortInd(sp)) =>
(* ends the current transaction ( if any)  *)
      update(sp, s) = s . (failed .= true) . (cl_cmt .. prepared .= false);
    IsTPBeginDialogueCnf(sp) =>
      update(sp, s) = s .
        (failed .= is_rejected(get_bdr(sp))) .
        (IndCnf .= true) .
        (DEReq .= true);
    IsTPEndDialogueReq(sp) =>
      update(sp, s) = s .
        (failed .= not(get_edc(sp))) .
        (ReqRsp .= true) .
        (DTReq .= get_edc(sp));
    IsTPEndDialogueInd(sp) =>
      update(sp, s) = s .
        (failed .= not(get_edc(sp))) .
        (cntrl += (s . UEReq and get_edc(sp))) .
        (IndCnf .= true) .
        (DTInd .= (get_edc(sp) and not(s . UEReq))) .
        (UEReq .= false);
    IsTPUErrorReq(sp) =>
      update(sp, s) = s .
        ( cntrl += (s . polarized and ((s . HSInd) or (s . HSGCInd) or (s . DTInd)))) .
        (ReqRsp .= true) .
        (HSInd .= false) .
        (HSGCInd .= false) .
        ( UEReq .= ( s .  polarized and not(s . cntrl) and
            not((s . HSInd) or (s . HSGCInd) or (s . DTInd)))
        ) .
        (DTInd .= false);
    IsTPUErrorInd(sp) =>
      update(sp, s) = s .
        ( cntrl &= not(s . polarized and ((s . HSReq) or (s . HSGCReq) or (s . DTReq)))
        ) .
        (IndCnf .= true) .
        (HSReq .= false) .
        (HSGCReq .= false) .
        ( UEInd .=
          ( (s . polarized) and
            (s . cntrl) and
            not((s . HSReq) or (s . HSGCReq) or (s . DTReq))
          )
        ) .
        (DTReq .= false);
    IsTPEndDialogueRsp(sp) or
    IsTPEndDialogueCnf(sp) or
    IsTPUAbortReq(sp) or
    IsTPUAbortInd(sp) or
    (IsTPPAbortInd(sp) and not(s . DEInd)) => update(sp, s) = s . (failed .= true);
    IsTPGrantControlReq(sp) =>
      update(sp, s) = s . (cntrl .= false) . (ReqRsp .= true) . (UEInd .= false);
    IsTPGrantControlInd(sp) =>
      update(sp, s) = s . (cntrl .= true) . (IndCnf .= true) . (UEReq .= false);
    IsTPHandshakeReq(sp) =>
      update(sp, s) = s . (ReqRsp .= true) . (HSReq .= true);
    IsTPHandshakeInd(sp) =>
      update(sp, s) = s .
        (cntrl += (s . UEReq)) .
        (IndCnf .= true) .
        (HSInd .= not(s . UEReq)) .
        (UEReq .= false);
    IsTPHandshakeRsp(sp) =>
      update(sp, s) = s . (ReqRsp .= true) . (HSInd .= false);
    IsTPHandshakeCnf(sp) =>
      update(sp, s) = s . (IndCnf .= true) . (HSReq .= false);
    IsTPHandshakeAndGrantControlReq(sp) =>
      update(sp, s) =
        s . (cntrl .= false) . (ReqRsp .= true) . (HSGCReq .= true);
    IsTPHandshakeAndGrantControlInd(sp) =>
      update(sp, s) = s .
        (cntrl .= true) .
        (IndCnf .= true) .
        (HSGCInd .= not(s . UEReq)) .
        (UEReq .= false);
    IsTPHandshakeAndGrantControlRsp(sp) =>
      update(sp, s) = s . (ReqRsp .= true) . (HSGCInd .= false);
    IsTPHandshakeAndGrantControlCnf(sp) =>
      update(sp, s) = s . (IndCnf .= true) . (HSGCReq .= false);
    IsTPBeginTransactionReq(sp) or IsTPBeginTransactionInd(sp) =>
      update(sp, s) = s .
        (cl_cmt .= true) .
        (ReqRsp_IndCnf(sp) .= true) .
        (Prp_sp .. HRInd .= false);
    IsTPDeferredEndDialogueReq(sp) or IsTPDeferredEndDialogueInd(sp) =>
      update(sp, s) = s . (ReqRsp_IndCnf(sp) .= true) . (DeferED .= true);
    IsTPDeferredGrantControlReq(sp) or IsTPDeferredGrantControlInd(sp) =>
      update(sp, s) = s . (ReqRsp_IndCnf(sp) .= true) . (DeferGC .= true);
    IsTPPrepareReq(sp) or IsTPPrepareInd(sp) =>
      update(sp, s) = s .
        (ReqRsp_IndCnf(sp) .= true) .
        (Prp_sp .= true) .
        (data_perm .= (s . polarized and get_bool(get_dp0(sp))));
    IsTPReadyInd(sp) => update(sp, s) = s . (IndCnf .= true) . (RdyInd .= true);
    IsTPCommitReq(sp) =>
      update(sp, s) = s . (ReqRsp .= true) . (CmtReq .= true);
    IsTPCommitInd(sp) =>
      update(sp, s) = s . (IndCnf .= true) . (CmtInd .= true);
    IsTPDoneReq(sp) =>
      update(sp, s) = s . (ReqRsp .= true) . (DoneReq .= true) . (AbRej .= false);
    IsTPCommitCompleteInd(sp) =>
      update(sp, s) = s .
        (failed += (s . DeferED)) .
        ( cntrl .=
          ( (s . DeferGC and (get_did(s) eq sup_dial))
            or
            (not(s . DeferGC) and (s . cntrl))
          )
        )
(*
DeferGrantControl Req/Ind on the dialogue then the subordinate has control
 *)
        .
        ( cl_cmt &= (not(s . unchained) and not(s . failed) and not(s . DeferED))
        ) .
        (Prp_sp .. prepared .= false);
    IsTPRollbackReq(sp) or IsTPRollbackInd(sp) =>
      update(sp, s) = s .
        (ReqRsp_IndCnf(sp) .= true) .
        (HSReq .. UEInd &= not(s . cl_cmt)) .
        (Rbk_sp .= true) .
        (Rbk_init .= true);
    IsTPRollbackCompleteInd(sp) =>
      update(sp, s) = s .
        ( cntrl .=
          ( (s . cl_cmt and (s . cntrl_trans))
            or
            (not(s . cl_cmt) and (s . cntrl))
```

```
         )) .
        (cl_cmt &= (not(s . unchained) and not(s . failed))) .
        (Prp_sp .. prepared .= false);
    IsTPHeuristicReportInd(sp) =>
      update(sp, s) = s . (IndCnf .= true) . (HRInd .= true);
    IsTPUAbortReq(sp) =>
(*      commitment but not transaction termination, i.e. rollback
initiating  *)
      special_update(sp, s) = s . (HSReq .. UEInd &= not(s . cl_cmt)) . (Rbk_init .= true);
    IsTPUAbortInd(sp) or IsTPPAbortInd(sp) or IsTPBeginDialogueCnf_rejected(sp)
    => (*  during commitment  *)
      special_update(sp, s) = s .
        (HSReq .. UEInd .= not(s . cl_cmt and get_rbk(sp))) .
        (Rbk_init += get_rbk(sp)) .
        (AbRej .= true);
    IsTPPrepareInd(sp) => (* sent on dialogue with superior   *)
      special_update(sp, s) = s . (prepared .= true);
    first_coord_commitment(superior, s) =
      s . (one_dial_cl_cmt .= true) . (prepared .= superior);
(*      if the node is the superior on the first dialogue with coordination
level commitment, then the node is the root of the new transaction, and so the
transaction is prepared *)
endtype
(* ------------------------------------------------------------------------
module service_info is ServicePrim, dialogue_channel_identifiers
/*service_info*/
    type service_info is
        info (DID: dc_id,
                BDC: tp_begin_dialogue_confirmation,
                BS: bit_string,
                NEXT_DID: dc_id)
    endtype

  function get_did (SI: service_info) : dc_id is select DID in SI enfunc
  function get_next_did (SI: service_info) : dc_id is select NEXT_DID in SI endfunc
  function get_bdc (SI: service_info) : tp_begin_dialogue_confirmation is
        select BDC in SI endfunc
  function _._ (SI: service_info, N: nat) : bool is get_bit (N, select BS in SI) endfunc
  function inc_next_did (SI: service_info) : service_info is
        SI.{NEXT_DID:=inc(select NEXT_DID in SI)} endfunc
/* range */
    type range is
         _.._ (N: nat, M:nat)
    endtype
/* assignment */
    type assignment is
        _.=_ (N: nat, B: bool)
    |   _&=_ (N: nat, B: bool)
    |   _+=_ (N: nat, B: bool)
    |   _.=_ (R: range, B: bool)
    |  _&=_ (R: range, B: bool)
    |  _.=_ (R: range, SI: service_info)
    endtype
```

```
function _._ (S: service_info, A: assignment) : service_info is
  case A in
      (N: nat .= B: bool) -> S.{BS:=set_bit (N, B, select BS in S}
  |   (N: nat &= B: bool) -> S . (N .= (S . N and B))
  |   (N: nat += B: bool) -> S . (N .= (S . N or B))
  |   ((N: nat .. M: nat) .= b) when (N eq M) -> S . (N .= B)
  |   ((N: nat .. M: nat) .= b) when (N lt M) -> S . (N .= B) . (succ(N) .. M .= B)
  |   ((N: nat .. M: nat) &= true) -> S
  |   ((N: nat .. M: nat) &= false) -> S . (N .. M .= false)
  |   ((N: nat .. M: nat) .= SI: service_info) when (N eq M) -> S  . (N .= (SI . N))
  |   ((N: nat .. M: nat) .= SI: service_info) when (N lt M) ->
       S . (N .= (SI . N)) . (succ(N) .. M .= SI)
  endcase
endfunc

function polarized : nat is 0 endfunc
function handshake : nat is succ(polarized) endfunc
function unchained : nat is succ(handshake) endfunc
function failed : nat is succ(unchained) endfunc
function cntrl : nat is succ(failed) endfunc
function cntrl_trans : nat is succ(cntrl) endfunc
function cl_cmt : nat is succ(cntrl_trans) endfunc
function ReqRsp : nat is succ(cl_cmt) endfunc
function IndCnf : nat is succ(ReqRsp) endfunc
function DEReq : nat is succ(IndCnf) endfunc
function DEInd : nat is succ(DEReq) endfunc
function HSReq : nat is succ(DEInd) endfunc
function HSInd : nat is succ(HSReq) endfunc
function HSGCReq : nat is succ(HSInd) endfunc
function HSGCInd : nat is succ(HSGCReq) endfunc
function UEReq : nat is succ(HSGCInd) endfunc
function UEInd : nat is succ(UEReq) endfunc
function DTReq : nat is succ(UEInd) endfunc
function DTInd : nat is succ(DTReq) endfunc
function Prp_sp : nat is succ(DTInd) endfunc
function data_perm : nat is succ(Prp_sp) endfunc
function RdyInd : nat is succ(data_perm) endfunc
function DeferED : nat is succ(RdyInd) endfunc
function DeferGC : nat is succ(DeferED) endfunc
function HRInd : nat is succ(DeferGC) endfunc
function one_dial_cl_cmt : nat is succ(HRInd) endfunc
function CmtReq : nat is succ(one_dial_cl_cmt) endfunc
function CmtInd : nat is succ(CmtReq) endfunc
function DoneReq : nat is succ(CmtInd) endfunc
function Rbk_sp : nat is succ(DoneReq) endfunc
function Rbk_init : nat is succ(Rbk_sp) endfunc
function AbRej : nat is succ(Rbk_init) endfunc
function Prepared : nat is succ(AbRej) endfunc
function ReqRsp_IndCnf (SP: ServicePrim) : nat is
      if IsReqRsp (SP) then ReqRsp
      elsif IsIndCnf (SP) then IndCnf endif
endfunc
function has_polarized (SP: ServicePrim) : bool is
      polarized_selected(get_sfu (SP)) endfunc
```

```
function has_shared (SP: ServicePrim) : bool is
      shared_selected(get_sfu (SP)) endfunc
function has_handshake (SP: ServicePrim) : bool is
      handshake_selected(get_sfu (SP)) endfunc
function has_commit (SP: ServicePrim) : bool is
      commit_selected(get_sfu (SP)) endfunc
function has_chained (SP: ServicePrim) : bool is
      chained_selected(get_sfu (SP)) endfunc
function has_unchained (SP: ServicePrim) : bool is
      unchained_selected(get_sfu (SP)) endfunc
function has_recovery (SP: ServicePrim) : bool is
      recovery_selected(get_sfu (SP)) endfunc
function dial_coord_commitment (SP: ServicePrim) : bool is
    has_chained (SP) or (has_unchained (SP) and get_bool(get_bt0 (SP))) endfunc
function IsMultiReq (SP: ServicePrim) : bool is
    IsTPCommitReq (SP) or IsTPRollbackReq (SP) or IsTPDoneReq (SP) endfunc
function IsMultiInd (SP: ServicePrim) : bool is
    IsTPCommitInd (SP) or
    IsTPCommitCompleteInd (SP) or
    IsTPRollbackInd (SP) or
    IsTPRollbackCompleteInd (SP) endfunc
function IsMulti (SP: ServicePrim) : bool is
      IsMultiReq (SP) or IsMultiInd (SP) endfunc


function info_sub(D: dc_id, SP: ServicePrim, ND: dc_id) : service_info is
    info( D, get_bdc (SP),
      has_polarized (SP)+(has_handshake (SP)+string (has_unchained (SP))),
      next_did) .
      (cntrl .= true) .
      (cntrl_trans .= true) .
      (cl_cmt .= dial_coord_commitment (SP)) .
      (ReqRsp .= true) .
      (DEReq .= is_always (get_bdc (SP))) endfunc
function info_sup(did, sp, next_did) =
    info( did, get_bdc(sp),
      has_polarized(SP) + (has_handshake(SP) + string(has_unchained(SP))),
      next_did) .
    (cl_cmt .= dial_coord_commitment(SP)) .
    (IndCnf .= true) .
    (DEInd .= is_always(get_bdc(SP))) endfunc
function  init_info (SP: ServicePrim) : service_info is
  if IsTPBeginDialogueReq (SP) and dial_coord_commitment (SP) then
      first_coord_commitment(true, info_sub (first_sub_dial, SP, inc (first_sub_dial)))
  elsif IsTPBeginDialogueReq (SP) and not (dial_coord_commitment (SP)) then
      info_sub (first_sub_dial, SP, inc (first_sub_dial))
  elsif IsTPBeginDialogueInd (SP) and dial_coord_commitment (SP) then
      first_coord_commitment (false, info_sup (sup_dial, SP, inc(sup_dial)))
  elsif IsTPBeginDialogueInd (SP) and not (dial_coord_commitment (SP))  then
      info_sup(sup_dial, SP, inc(sup_dial))
  endif
endfunc

function new_info (SP: ServicePrim, SI: service_info) : service_info is
```

```
    if IsTPBeginDialogueReq (SP) then
      info_sub(get_next_did (SI), sp, did_absent).(one_dial_cl_cmt .. prepared .= SI)
    endif
  endfunc

  function update (sp: ServicePrim, s: service_info) : service_info is
    if  IsTPDataReq(sp) or
        IsTPDataInd(sp) or
        IsTPRequestControlReq(sp) or
        IsTPRequestControlInd(sp)
        then  s . (ReqRsp_IndCnf(sp) .= true)
    elsif IsTPBeginDialogueReq(sp)
        then inc_next_did(s) . (ReqRsp .= true)
    elsif IsTPBeginDialogueRsp_accepted(sp)
        then s . (ReqRsp .= true) . (DEInd .= false)
    elsif IsTPBeginDialogueRsp_rejected(sp) or (s . DTInd and IsTPPAbortInd(sp))
        then s . (failed .= true) . (cl_cmt .. prepared .= false)
    elsif IsTPBeginDialogueCnf(sp)
        then  s . (failed .= is_rejected(get_bdr(sp))) .
                (IndCnf .= true) . (DEReq .= true)
    elsif IsTPEndDialogueReq(sp)
        then s . (failed .= not(get_edc(sp))) .
                (ReqRsp .= true) . (DTReq .= get_edc(sp))
    elsif IsTPEndDialogueInd(sp)
        then s .
                (failed .= not(get_edc(sp))) .
                (cntrl += (s . UEReq and get_edc(sp))) .
                (IndCnf .= true) .
                (DTInd .= (get_edc(sp) and not(s . UEReq))) .
                (UEReq .= false)
    elsif IsTPUErrorReq(sp)
        then s .
        ( cntrl += (s . polarized and ((s . HSInd) or (s . HSGCInd) or (s . DTInd)))) .
        (ReqRsp .= true) .
        (HSInd .= false) .
        (HSGCInd .= false) .
        ( UEReq .= ( s .  polarized and not(s . cntrl) and
            not((s . HSInd) or (s . HSGCInd) or (s . DTInd)))
        ) .
        (DTInd .= false)
    elsif IsTPUErrorInd(sp)
        then s .
        ( cntrl &= not(s . polarized and ((s . HSReq) or (s . HSGCReq) or (s . DTReq)))
        ) .
        (IndCnf .= true) .
        (HSReq .= false) .
        (HSGCReq .= false) .
        ( UEInd .=
          ( (s . polarized) and
            (s . cntrl) and
            not((s . HSReq) or (s . HSGCReq) or (s . DTReq))
          )
        ) .
        (DTReq .= false)
```

```
elsif IsTPEndDialogueRsp(sp) or
      IsTPEndDialogueCnf(sp) or
      IsTPUAbortReq(sp) or
      IsTPUAbortInd(sp) or
      (IsTPPAbortInd(sp) and not(s . DEInd))
    then s . (failed .= true)
elsif IsTPGrantControlReq(sp)
    then s . (cntrl .= false) . (ReqRsp .= true) . (UEInd .= false)
elsif IsTPGrantControlInd(sp)
    then s . (cntrl .= true) . (IndCnf .= true) . (UEReq .= false)
elsif IsTPHandshakeReq(sp)
    then s . (ReqRsp .= true) . (HSReq .= true)
elsif IsTPHandshakeInd(sp)
    then    s .
            (cntrl += (s . UEReq)) .
            (IndCnf .= true) .
            (HSInd .= not(s . UEReq)) .
            (UEReq .= false)
elsif IsTPHandshakeRsp(sp)
    then s . (ReqRsp .= true) . (HSInd .= false)
elsif IsTPHandshakeCnf(sp)
    then s . (IndCnf .= true) . (HSReq .= false)
elsif IsTPHandshakeAndGrantControlReq(sp) =>
    then  s . (cntrl .= false) . (ReqRsp .= true) . (HSGCReq .= true)
elsif IsTPHandshakeAndGrantControlInd(sp)
    then  s .
    (cntrl .= true) .
    (IndCnf .= true) .
    (HSGCInd .= not(s . UEReq)) .
    (UEReq .= false)
elsif IsTPHandshakeAndGrantControlRsp(sp)
    then  s . (ReqRsp .= true) . (HSGCInd .= false)
elsif IsTPHandshakeAndGrantControlCnf(sp)
    then  s . (IndCnf .= true) . (HSGCReq .= false)
elsif IsTPBeginTransactionReq(sp) or IsTPBeginTransactionInd(sp)
    then  s .
    (cl_cmt .= true) .
    (ReqRsp_IndCnf(sp) .= true) .
    (Prp_sp .. HRInd .= false)
elsif IsTPDeferredEndDialogueReq(sp) or IsTPDeferredEndDialogueInd(sp)
    then  s . (ReqRsp_IndCnf(sp) .= true) . (DeferED .= true)
elsif IsTPDeferredGrantControlReq(sp) or IsTPDeferredGrantControlInd(sp)
    then  s . (ReqRsp_IndCnf(sp) .= true) . (DeferGC .= true)
elsif IsTPPrepareReq(sp) or IsTPPrepareInd(sp)
    then  s .
    (ReqRsp_IndCnf(sp) .= true) .
    (Prp_sp .= true) .
    (data_perm .= (s . polarized and get_bool(get_dp0(sp))))
elsif IsTPReadyInd(sp)      then  s . (IndCnf .= true) . (RdyInd .= true)
elsif IsTPCommitReq(sp)
    then  s . (ReqRsp .= true) . (CmtReq .= true)
elsif IsTPCommitInd(sp)
    then  s . (IndCnf .= true) . (CmtInd .= true)
elsif IsTPDoneReq(sp)
```

```
        then  s . (ReqRsp .= true) . (DoneReq .= true) . (AbRej .= false)
    elsif IsTPCommitCompleteInd(sp)
        then  s .
        (failed += (s . DeferED)) .
        ( cntrl .=
          ( (s . DeferGC and (get_did(s) eq sup_dial))
            or
            (not(s . DeferGC) and (s . cntrl))
          )
        )
        .
        ( cl_cmt &= (not(s . unchained) and not(s . failed) and not(s . DeferED))
        ) .
        (Prp_sp .. prepared .= false)
    elsif IsTPRollbackReq(sp) or IsTPRollbackInd(sp)
        then  s .
        (ReqRsp_IndCnf(sp) .= true) .
        (HSReq .. UEInd &= not(s . cl_cmt)) .
        (Rbk_sp .= true) .
        (Rbk_init .= true)
    elsif IsTPRollbackCompleteInd(sp)
        then  s .
        ( cntrl .=
          ( (s . cl_cmt and (s . cntrl_trans))
            or
            (not(s . cl_cmt) and (s . cntrl))
          )) .
        (cl_cmt &= (not(s . unchained) and not(s . failed))) .
        (Prp_sp .. prepared .= false)
    elsif IsTPHeuristicReportInd(sp)
        then  s . (IndCnf .= true) . (HRInd .= true)
  endfunc


  function special_update (sp: ServicePrim, s: service_info) : service_info is
    if IsTPUAbortReq(sp)
        then s . (HSReq .. UEInd &= not(s . cl_cmt)) . (Rbk_init .= true)
    elsif IsTPUAbortInd(sp) or IsTPPAbortInd(sp) or IsTPBeginDialogueCnf_rejected(sp)
        then s .
        (HSReq .. UEInd .= not(s . cl_cmt and get_rbk(sp))) .
        (Rbk_init += get_rbk(sp)) .
        (AbRej .= true)
    elsif IsTPPrepareInd(sp)
        then s . (prepared .= true)
    endif
  endfunc

  function first_coord_commitment (B: bool, s: service_info) : service_info is
      s . (one_dial_cl_cmt .= true) . (prepared .= B) endfunc
endmod
========================================================================== *)

(*
 *      H.5.11  Implementation Dependent Values
 *)
type local_values is parameters
opns
  my_functional_units (*! constructor *) : -> functional_units
  my_aet (*! constructor *) : -> AE_title
  my_addr (*! constructor *) : -> address
  my_version : -> protocol_version
eqns
  ofsort protocol_version
    my_version = protocol_version(string(true));
(*      protocol version 1 only  *)
endtype
(* --------------------------------------------------------------------------
module local_values is parameters
/* moved in TP_parameters module
  my_functional_units : -> functional_units */
/* moved in addressing module
  my_aet : -> AE_title
  my_addr : -> address */

  function my_version : protocol_version is protocol_version (string(true)) endfunc
endmod
========================================================================== *)


endspec
```

Chapter 4

ODP Application (1)

4.1 Comments

We only examine the data part of the ODP "metamovie" application, and we keep the behaviour part unchanged.

From our analysis of the formal description of this ODP application, we can make the following remarks:

- We notice that type "`ident_types`" has two different equality relations: the syntactical equality and a user-defined one.
- There are some sorts without constructors (e.g., the `frame` sort of `frame_type` type). These sorts could be considered to be external.
- For all types, selector functions are defined. However, tester functions are not used.

4.2 Formal description in both LOTOS and E-LOTOS

```
(*
Author: Andreas Vogel  July, 1993
Modified by: Mihaela Sighireanu November 11995
        - identified constructors, corrected equations, and provided a E-LOTOS translation
*)

specification metamovie[ req, resp ]: noexit

library Boolean endlib

type ident_type is Boolean
        sorts   ident
        opns    server (*! constructor *) ,
                        server_stream_interface (*! constructor *) ,
                        server_operational_interface (*! constructor *) ,
                client (*! constructor *) ,
                        client_stream_interface (*! constructor *) ,
                        client_operational_interface (*! constructor *) ,
                infrastructure (*! constructor *) ,
                activate_interface (*! constructor *),
                deactivate_interface (*! constructor *),
                instantiate (*! constructor *),
                start (*! constructor *),
                halt (*! constructor *),
                destroy (*! constructor *),
                dir (*! constructor *),
                change (*! constructor *),
                create (*! constructor *),
                create_stream (*! constructor *),
                x (*! constructor *),
                y (*! constructor *),
                z (*! constructor *),
                p2p_stream (*! constructor *),
                broadcast_stream (*! constructor *) : -> ident
                nil_if (*! constructor *),
                p_op_if (*! constructor *),
                c_op_if (*! constructor *),
                p_stream_if (*! constructor *),
                c_stream_if (*! constructor *),
                s (*! constructor *),
                bi (*! constructor *) : -> ident

                _eq_: ident, ident -> bool
                _ne_: ident, ident -> bool

        eqns forall x: ident
        ofsort bool
(* originel : infrastructure ne infrastructure = false; *)
(* modified *)
        x = x => x eq x = true;
        x ne x = not (x eq x);
endtype (* ident_type *)
```

```
(* ------------------------------------------------------------------------
        module ident_type is Boolean
                 type   ident is
                        server
                        | server_stream_interface
                        | server_operational_interface
                        | client
                        | client_stream_interface
                        | client_operational_interface
                        | infrastructure
                        | activate_interface
                        | deactivate_interface
                        | instantiate
                        | start
                        | halt
                        | destroy
                        | dir
                        | change
                        | create
                        | create_stream
                        | x
                        | y
                        | z
                        | p2p_stream
                        | broadcast_stream
                        | nil_if
                        | p_op_if
                        | c_op_if
                        | p_stream_if
                        | c_stream_if
                        | s
                        | bi
                        | par2ident0
                endtype
        endmod
============================================================================ *)

type dir_type is
        sorts
                dir
        opns
                md (*! constructor *) : -> dir
endtype (* dir_type *)
(* ------------------------------------------------------------------------
        module dir_type is
                type dir is
                       md
                endtype
        endmod
============================================================================ *)

type parameter_type is
        ident_type, dir_type
        sorts
```

```
		parameter, plist
	opns
		infrastructure_failure (*! constructor *) : -> parameter
		dir2par (*! constructor *) : dir -> parameter
		ident2par0 (*! constructor *) : ident -> parameter
		par2ident0 (*! constructor *) : parameter -> ident

		par2dir: parameter -> dir
		ident2par: ident -> parameter
		par2ident: parameter -> ident

		o (*! constructor *) : -> plist
		_._ (*! constructor *) : parameter, plist -> plist
		head: plist -> parameter
		tail: plist -> plist
	eqns forall p: parameter, id: ident, pl: plist
	ofsort dir
		par2dir( dir2par(md) ) = md;
	ofsort ident
		par2ident( ident2par0(id) ) = id;
	ofsort parameter
		ident2par( par2ident0(p) ) = p;

	ofsort parameter
		head( o )    = infrastructure_failure;
		head( p.pl ) = p;
	ofsort plist
		tail( o )    = o;
		tail( p.pl ) = pl;

endtype (* parameter_type *)
(* -----------------------------------------------------------------------
	module parameter_type is ident_type, dir_type
		type parameter is
			  infrastructure_failure
			| dir2par(d:dir)
			| ident2par(i:ident)
		endtype
		function par2dir(P:parameter) is select d in P endfunc
		function par2ident(P:parameter) is select i in P endfunc
		type plist is
			  o
			| _._ (p:parameter, pl:plist)
		endtype
		function head (Pl:plist) : parameter select p in PI endfunc
		function tail(Pl: plist) : plist select pl in Pl endfunc
	endmod
======================================================================= *)

type video_qos_type is
	sorts
		color, frequence
	opns
		bw (*! constructor *),
                pal (*! constructor *),
                secam (*! constructor *),
                ntsc (*! constructor *),
                hdtv (*! constructor *) : -> color
                normal (*! constructor *),
                slow_motion (*! constructor *) : -> frequence
endtype (* video_qos_type *)
(* ----------------------------------------------------------------------------
        module video_qos_type is
                type color is
                          bw
                        | pal
                        | secam
                        | ntsc
                        | hdtv
                endtype
                type frequence is
                          normal
                        | slow_motion
                endtype
        endmod
============================================================================ *)

type frame_type is
        sorts
                frame
(* added *)
        opns
                frame (*! constructor *) : -> frame
endtype (* frame_type *)
(* ----------------------------------------------------------------------------
        module frame_type is
                type frame is
                        frame
                endtype
        endmod
============================================================================ *)

behaviour

hide announcement, invocation, termination in

object_space[ req, resp, announcement, invocation, termination ]
|[ announcement, invocation, termination ]|
infrastructure[ announcement, invocation, termination ]

where

process object_space[ req, resp, announcement, invocation, termination ]:
        noexit :=

        hide get, put in
        choice b: ident [] binder[ put, get, announcement, invocation, termination ]( b )
        |[ get, put ]|
        ( server[ put, invocation, termination ]( p_op_if, md )
        |||
        client[ get, invocation, termination ]( p_op_if, c_op_if, c_stream_if ))
where

process server[put, invocation, termination]
        ( op_if: ident, movie_dir: dir ): noexit :=

        hide control in
        stream_interface_manager[ control, put ]
        |[ control ]|
        server_operational_interface[invocation, termination, control, put]
                (op_if, movie_dir)
where
        process server_operational_interface[invocation, termination, control,
                        put](interface: ident, movie_dir: dir): noexit :=

                start_operation[invocation, termination, control, put]
                        (interface)
                |||
                stop_operation[invocation, termination, control](interface)
                |||
                dir_operation[invocation, termination, control]
                        (interface, movie_dir)
                |||
                change_operation[invocation, termination, control](interface)
        where
                process start_operation[invocation, termination, control, put]
                        ( self: ident ): noexit :=
                        invocation !server ?client_op_if: ident !self
                                   !start ?p: plist;
                        (let client_stream_if: ident = par2ident(head(p)) in
                        hide x in x ?server_stream_if: ident;
                        control !server_stream_if !create;
                        (choice b: ident []
                        invocation !client !self !b !create_stream
                                !ident2par(server_stream_if).(ident2par(client_stream_if).o);
                        termination !client !self !b !create_stream ?p: plist;
                        ( let   stream_op_if: ident = par2ident(head(p)),
                                source_if: ident = par2ident(head(tail(p))),
                                sink_if: ident = par2ident(head(tail(tail(p))))
                        in
                        invocation !client !self !stream_op_if !start
                                !ident2par(source_if).o;
                        termination !client !self !stream_op_if !start
                                ?r1: plist;
                        invocation !client !self !stream_op_if !start
                                !ident2par(sink_if).o;
                        termination !client !self !stream_op_if !start
                                ?r2: plist;
                        control !server_stream_if !start
                                !ident2par(source_if).o;
                        termination !server !client_op_if !self !start
                        !ident2par(server_stream_if).(ident2par(sink_if).o);
                        start_operation[invocation, termination,
                                control, put](self)
                        ) (* fin let *)
                        ) (* fin choice *)
                        ) (* fin let *)
                endproc (* start_operation *)

                process stop_operation[invocation, termination, control]
                        ( self: ident ): noexit :=
                        invocation !server ?client: ident !self
                                   !halt ?p: plist;
                        control !par2ident(head(p)) !halt !p;
                        termination !server !client !self !halt !p;
                        stop_operation[invocation, termination,
                                control](self )
                endproc (* stop_operation *)

                process dir_operation[invocation, termination, control]
                        ( self: ident, movie_dir: dir ): noexit :=
                        invocation !server ?client: ident !self
                                   !dir ?p: plist;
                        termination !server !client !self !dir
                                !dir2par(movie_dir).o;
                        dir_operation[invocation, termination,
                                control](self, movie_dir)
                endproc (* dir_operation *)

                process change_operation[invocation, termination, control]
                        ( self: ident ): noexit :=
                        invocation !server ?client: ident !self
                                   !change ?p: plist;
                        control !par2ident(head(p)) !change !p;
                        termination !server !client !self !change !p;
                        change_operation[invocation, termination,
                                control](self )
                endproc (* change_operation *)
        endproc (* server_operation_interface *)

        process stream_interface_manager[ control, put ]: noexit :=
                control ?if: ident !create;
                ( server_stream_interface[ control, put ]( if )
                |||
                stream_interface_manager[ control, put ] )
        endproc (* stream_interface_manager *)

        process server_stream_interface[ control, put ]
                ( self: ident): noexit :=

                control !self !start ?p: plist;
                ( stream[ put ](self, p)
                        [>      control !self !change ?p: plist;
                        stream[ put ](self, p)
                        [>      control !self !halt ?p: plist;
                        server_stream_interface[ control, put ](self)
                [>      control !self !destroy ?p: plist;
                        stop )
        where
                process stream[ put ](self: ident, p: plist): noexit :=
                        put !self !par2ident(head(p)) ?f: frame;
                        stream[ put ](self, p)
                endproc (* stream *)
        endproc (* server_stream_interface *)
endproc (* server *)

process client[get, invocation, termination]
        ( server_op_if ,op_if, stream_if: ident ): noexit :=

        hide control in
        client_stream_interface[control, get ](stream_if)
        |[ control ]|
        client_operational_interface[invocation, termination, control ]
                ( server_op_if, op_if, stream_if, nil_if)
where
        process client_operational_interface[invocation, termination, control]
                ( server_op_if, self, client_stream_if, server_stream_if:
                ident): noexit :=
                start_operation[invocation, termination, control]
                        (server_op_if, self, client_stream_if,
                        server_stream_if)
                []
                stop_operation[invocation, termination, control]
                        (server_op_if, self, client_stream_if,
                        server_stream_if)
                []
                dir_operation[invocation, termination, control]
                        (server_op_if, self, client_stream_if,
                        server_stream_if)
                []
                change_operation[invocation, termination, control]
                        (server_op_if, self, client_stream_if,
                        server_stream_if)
        where
                process start_operation[invocation, termination, control]
                        ( server_op_if, self, client_stream_if,
                        server_stream_if: ident ): noexit :=
                        invocation !client !self !server_op_if
                                   !start !ident2par(client_stream_if).o;
                        termination !client !self !server_op_if !start
                                ?res: plist;
                        ( let server_stream_if: ident = par2ident(head(res)),
                                sink_if: ident = par2ident(head(tail(res)))
                        in
                        control !start !ident2par(sink_if).o;
                        client_operational_interface[invocation, termination,
                        control ]( server_op_if, self, client_stream_if,
                         par2ident(head(res)) ) )
                endproc (* start_operation *)
                process stop_operation[invocation, termination, control]
                        ( server_op_if, self, client_stream_if,
                        server_stream_if: ident ): noexit :=
                        invocation !client !self !server_op_if !halt
                                !ident2par(server_stream_if).o;
                        control !halt !o;
                        termination !client !self !server_op_if !halt ?p: plist;
                        stop_operation[invocation, termination, control]
                                ( server_op_if, self, client_stream_if,
                                server_stream_if )
                endproc (* stop_operation *)

                process dir_operation[invocation, termination, control]
                        ( server_op_if, self, client_stream_if,
                        server_stream_if: ident ): noexit :=
                        invocation !client !self !server_op_if
                                   !dir ?p: plist;
                        termination !client !self !server_op_if !dir ?p: plist;
                        stop_operation[invocation, termination, control]
                                ( server_op_if, self, client_stream_if,
                                server_stream_if )
                endproc (* dir_operation *)

                process change_operation[invocation, termination, control]
                        ( server_op_if, self, client_stream_if,
                        server_stream_if: ident ): noexit :=
                        invocation !client !self !server_op_if
                                   !change ?p: plist;
                        control !change !p;
                        termination !client !self !server_op_if !change
                                ?p: plist;
                        change_operation[invocation, termination, control]
                                ( server_op_if, self, client_stream_if,
                                server_stream_if )
                endproc (* change_operation *)
        endproc (* client_operation_interface *)

        process client_stream_interface[ control, get ]
                ( interface: ident): noexit :=
                control !start ?p: plist;
                ( stream[ get ](interface, p)
                        [> control !halt ?p: plist;
                        client_stream_interface[ control, get ]( interface ) )
        where
                process stream[ get ]
                        (interface: ident, p: plist): noexit :=
                        get !par2ident(head(p)) !interface ?f: frame;
                        stream[ get ](interface, p)
                endproc (* stream *)
        endproc (* client_stream_interface *)
endproc (* client *)

process stream_object[ put, get, announcement, invocation, termination ]
                ( op_if, source_if, sink_if, source, sink: ident ): noexit :=
        interface_manager[put, get, invocation, termination]
                ( op_if, source_if, sink_if, source, sink )
where
        process interface_manager[put, get, invocation, termination]
                ( op_if, source_if, sink_if, source, sink: ident ): noexit :=

                hide control, buffer in
                ( source_site_stream_interface[control, put, buffer ]
                        (source_if, source)
                |[buffer]|
                  sink_site_stream_interface[control, get, buffer ]
                        ( sink_if, sink) )
                |[ control ]|
                stream_operational_interface[invocation, termination,
                        control ](op_if, source_if, sink_if)
        endproc (* interface_manager *)

        process stream_operational_interface[invocation, termination,
                        control ]
                ( self, source_if, sink_if: ident): noexit :=
                start_operation[invocation, termination,
                        control ](self, source_if, sink_if)
                |||
                stop_operation[invocation, termination,
                        control ](self, source_if, sink_if)
        where
                process start_operation[invocation, termination, control]
                        ( self, source_if, sink_if: ident ): noexit :=
                        invocation !server ?client: ident !self !start ?p: plist;
                        control !par2ident(head(p)) !start !tail(p);
                        termination !server !client !self !start !p;
                        stream_operational_interface[invocation, termination,
                                control](self, source_if, sink_if )
                endproc (* start_operation *)

                process stop_operation[invocation, termination, control]
                        ( self, source_if, sink_if: ident ): noexit :=
                        invocation !server ?client: ident !self !halt ?p: plist;
                        control !par2ident(head(p)) !halt !tail(p);
                        termination !server !client !self !halt !p;
                        stream_operational_interface[invocation, termination,
                                control](self, source_if, sink_if )
                endproc (* stop_operation *)
        endproc (* stream_operational_interface *)

        process source_site_stream_interface[control, put, buffer ]
                (self, source: ident): noexit :=
                control !self !start ?p: plist;
                ( stream[ put, buffer ](self, source)
                [> control !self !halt ?p: plist;
                source_site_stream_interface[ control, put, buffer ]
                        ( self, source ) )
        where
```

```
                process stream[ put, buffer ]
                        (self, source: ident): noexit :=
                        put !source !self ?f:frame;
                        buffer !f;
                        stream[ put, buffer ] (self, source)
                endproc (* stream *)
        endproc (* source_site_stream_interface *)

        process sink_site_stream_interface[control, get, buffer ]
                (self, sink: ident): noexit :=
                control !self !start ?p: plist;
                ( stream[ get, buffer ](self, sink)
                [> control !self !halt ?p: plist;
                sink_site_stream_interface[ control, get, buffer ]
                        ( self, sink ) )
        where
                process stream[ get, buffer ]
                        (self, sink: ident): noexit :=
                        buffer ?f:frame;
(*      in case of multicasting, the sink parameter should removed,
        then n clients can read from the stream, synchroniously:
                        get !self !f;
*)
                        get !self !sink !f;
                        stream[ get, buffer ](self, sink)
                endproc (* stream *)
        endproc (* sink_site_stream_interface *)
endproc (* stream_object *)

process binder[ put, get, announcement, invocation, termination ]
                ( self: ident ): noexit :=
(*
        hide create_ident in
*)
        invocation !server ?sender: ident !self !create_stream ?p: plist;
(*
        ( create_ident ?op_if: ident ?source_if: ident ?sink_if: ident;
*)
        ( let op_if: ident = x, source_if: ident = y, sink_if: ident = z in
        ( ( stream_object[ put, get, announcement, invocation, termination ]
                ( op_if, source_if, sink_if, par2ident(head( p )),
                        par2ident(head(tail(p)))) )
        |||
          ( termination !server !sender !self !create_stream
                !ident2par(op_if).(ident2par(source_if).(ident2par(sink_if).o));
            stop ))
        |||
        binder[ put, get, announcement, invocation, termination ]( self ))
endproc (* binder *)
endproc (* object_space *)

process infrastructure[ announcement, invocation, termination ]: noexit :=

        announcement !client ?sender: ident ?receiver: ident ?p: plist;
        ( ( announcement !server !sender !receiver !p;
            stop )
        |||
        infrastructure[ announcement, invocation, termination ] )
        []
        invocation !client ?sender: ident ?receiver: ident ?op: ident ?p: plist;
        ((( invocation !server !sender !receiver !op !p;
            termination !server !sender !receiver !op ?p: plist;
            termination !client !sender !receiver !op !p;
            stop )
            [> termination !sender !receiver !op !infrastructure_failure;
               stop)
        |||
        infrastructure[ announcement, invocation, termination ] )
endproc (* infrastructure *)
endspec
```

Chapter 5

ODP Application (2)

5.1 Comments

We only examine the data part of the ODP "video service" application, and we keep the behaviour part unchanged.

From our analysis of the formal description of this ODP application, we can make the following remarks:

- First, all the comments made for the ODP "metamovie" application also apply to this chapter.
- Mutually recursive types should be declared in the same module
- Most of the types in this description are enumerated types. For all enumerated types, it might be useful to have a predefined "`ord`" function (as in Pascal) that maps every constructor onto a natural number.
- Using the proposed E-LOTOS extensions, the "`remove`" functions could be specified in many ways:

  1. using pattern-matching:

     ```
     function remove (id: ident, L: identList) : identList is
       case L in
           o -> o
         | (ID:=id1: ident).(IDL:=idlist: identList) when id equ id1 ->
              idlist
         | (ID:=id1: ident).(IDL:=idlist: identList) when id neq id1 ->
              id1.remove(id,idlist)
       endcase
     endfunc
     ```

  2. using a "select" statement:

     ```
     function remove (id: ident, L: identList) : identList is
       case L in
           o -> o
         | any identList when id equ (select ID in L) ->
     ```

```
          select IDL in L
       | any identList when id neq (select ID in L) ->
          (select ID inL).remove(id, select ID in L)
       endcase
     endfunc
```

3. or even:

```
     function remove (id: ident, L: identList) : identList is
      case L in
          o -> o
      | any identList -> let id1 = select ID in L, idlist = select IDL in L in
                 if id equ id1 then idlist
                 else id1.remove(id,idlist)
       endcase
     endfunc
```

4. or even:

```
     function remove (id: ident, L: identList) : identList is
      case L in
          o -> o
      | (ID:=id: ident).(IDL:= idlist: identList) ->  idlist
      | any identlist -> (select ID in L).remove(id, select ID in L)
      endcase
     endfunc
```

5.2 Formal description in both LOTOS and E-LOTOS

```
(*
Author: Frank Koch 31. Januar 1994
Modified by: Mihaela Sighireanu November 1995
        - identified constructors, corrected equations, and provided a E-LOTOS translation
*)

specification video_service : noexit

(* added NATURALNUMBER; defined Integer like NATURALNUMBER renamed *)
library Boolean, Integer, NATURALNUMBER  endlib

type  identifier_type  is Boolean
 sorts ident, identList
 opns
        blank   (*! constructor *):     -> ident
        o       (*! constructor *):     -> identList
        _._     (*! constructor *): ident, identList -> identList
        remove  : ident, identList -> identList
        head    : identList -> ident
        tail    : identList -> identList
        empty   : identList -> bool
        inlist  : ident, identList -> bool
        select  : ident, identList -> ident
        _equ_   : ident, ident -> bool
        _neq_   : ident, ident -> bool
 eqns forall id, id1: ident, idlist: identList
     ofsort ident
        head( id.idlist ) = id;
        select( id,o ) = blank;
        id equ id1 => select( id,id1.idlist ) = id1;
        id neq id1 => select( id,id1.idlist ) = select( id,idlist );
     ofsort identList
        tail( id.idlist ) = idlist;
        remove( id,o ) = o;
        id equ id1 => remove( id,id1.idlist ) = idlist;
        id neq id1 => remove( id,id1.idlist ) = id1.remove( id,idlist );
     ofsort bool
        empty( id.o ) = false;
        empty( o ) = true;
        empty( id.idlist ) = false;
        id equ id = true;
        id neq id1 = not( id equ id1 );
        inlist( id,o ) = false;
        inlist( id, id1.idlist ) = (id equ id1) or inlist( id, idlist );
endtype (* identifier_type *)
(* -------------------------------------------------------------------------
        module  identifier_module  is Boolean
        type    ident is
                 blank
        endtype
        type    identList is
                  o
                | _._ (ID : ident, IDL: identList)
        endtype
        function remove (ID: ident, L: identList) : identList is
          case L in
                  o -> o
                | any identList -> let id1 = select ID in L, idlist = select IDL in L in
                      if id equ id1 then idlist
                      else id1.remove(id,idlist)
          endcase
        endfunc
        function head (L: identList) : ident is select ID in L endfunc
        function tail (L: identList) : identList is select L in L endfunc
        function empty  (L: identList) : bool is L match empty endfunc
        function inlist (id: ident, L: identList) : bool is
          case L in
                  o -> false
                | any identList -> (id equ (select ID in L)) or inlist(id, select IDL in L)
          endcase
        endfunc
        function select (id: ident, L: identList) : ident is
          case L in
                  o -> blank
                | any identList -> let id1 = select ID in L, idlist = select IDL in L in
                      if id equ id1 then id
                      else select(id,idlist)
          endcase
        endfunc
        endmod
============================================================================ *)

type video_qos_type is
        sorts
                color, frequence
        opns
                bw (*! constructor *),
                pal (*! constructor *),
                secam (*! constructor *),
                ntsc (*! constructor *),
                hdtv (*! constructor *)         : -> color
                normal (*! constructor *),
                slow_motion (*! constructor *)          : -> frequence
endtype (* video_qos_type *)
(* ---------------------------------------------------------------------------
        module video_qos_type is
        type    color is
                  bw
                | pal
                | secam
                | ntsc
                | hdtv
        endtype
        type    frequence is
                  normal
                | slow_motion
        endtype
        endmod
============================================================================ *)

type   Failure_Types   is
  sorts DsError, CrError, InfraError
         opns
(* typical error messages for some terminations of "start" operations: *)
        no_such_video  (*! constructor *) : -> CrError
(* typical error messages for other operation terminations : *)
         no_such_interface (*! constructor *) : -> DsError
         non_authorizised_user (*! constructor *) : -> DsError
         bad_interface_id (*! constructor *) : -> DsError
endtype (* Failure_Types *)
(* ---------------------------------------------------------------------------
        module   Failure_Types   is
        type    DsError is
                   no_such_interface
                 | non_authorizised_user
                 | bad_interface_id
        endtype
        type    CrError is
                no_such_video
        endtype
        type    InfraError is
        endtype
        endmod
=========================================================================== *)

type   TimeFormat_Type   is Integer
 sorts h_TimeFormat, sec_TimeFormat
 opns
   _min_  (*! constructor *)    : Int, Int -> sec_TimeFormat
   _h_  (*! constructor *)      : Int, sec_TimeFormat   -> h_TimeFormat
endtype (* TimeFormat_Type *)
(* --------------------------------------------------------------------------
        module  TimeFormat_Type   is Integer
        type    h_TimeFormat is
           _h_ (Int, sec_TimeFormat)
        endtype
        type    sec_TimeFormat is
                _min_  (Int, Int)
        endtype
        endmod
=========================================================================== *)

type  video_titel_type  is Boolean, Integer
 sorts titel, tList
 opns
         a  (*! constructor *)  : -> tList
        _._ (*! constructor *)  : titel, tList -> tList
        head    : tList -> titel
        tail    : tList -> tList
        empty   : tList -> bool
        in_list : titel, tList -> bool
        _teq_   : titel, titel -> bool
        _tne_   : titel, titel -> bool
        map2int : titel -> int

        Dances_with_wolves (*! constructor *)   : -> titel
        Jurassic_park    (*! constructor *) : -> titel
        The_piano (*! constructor *)    : -> titel
        JFK (*! constructor *)          : -> titel
        Moonwalker (*! constructor *)   : -> titel
        no_titel (*! constructor *)     : -> titel
 eqns forall t,s: titel, tl: tList
        ofsort titel
                head( t.tl ) = t;
        ofsort tList
                tail( t.tl ) = tl;
        ofsort int
                map2int( no_titel ) = 0;
                map2int( JFK ) = s(0);
                map2int( The_piano ) = s(s(0));
                map2int( Jurassic_park ) = s(s(s(0)));
                map2int( Dances_with_wolves ) = s(s(s(s(0))));
                map2int( Moonwalker ) = s(s(s(s(s(0)))));
        ofsort bool
                t teq s = map2int(t) eq map2int(s);
                t tne s = not( t teq s );
                empty( t.a  ) = false;
                empty( a ) = true;
                empty( t.tl ) = false;
                in_list( t,a  ) = false;
                in_list( t,s.tl ) = (s teq t) or in_list( t,tl );
endtype (* video_titel_type *)
(* ---------------------------------------------------------------------------
        module  video_titel_type  is Boolean, Integer
        types titel is
                  Dances_with_wolves
                | Jurassic_park
                | The_piano
                | JFK
                | Moonwalker
                | no_titel
        endtype
        type tList is
                 a
                | _._ (T: titel, TL: tList)
        endtype
        function head (L: tList) : titel is select T in L endfunc
        function tail (L: tList) : tList is select TL in L endfunc
        function empty (L: tList) : bool is L match empty endfunc
        function in_list (T: titel, TL: tList) : bool is
          case TL in
                  a -> false
                | any tList -> ((select T in TL) eq T) or in_list (T, select TL in TL)
          endcase
        endfunc
        function map2int (T: titel) : int is
          case T in
                  no_titel -> 0
                | JFK -> 1
                | The_piano -> 2
                | Jurassic_park -> 3
                | Dances_with_wolves -> 4
                | Moonwalker -> 5
          endcase
        endtype
        endmod
=========================================================================== *)

type parameter_type is
        identifier_type,    TimeFormat_Type,   Failure_Types, video_titel_type
        sorts
                parameter, plist
        opns
                CrErr2par (*! constructor *) : CrError -> parameter
                par2CrErr: parameter -> CrError
                DsErr2par (*! constructor *) : DsError -> parameter
                par2DsErr: parameter -> DsError
                InfraErr2par (*! constructor *) : InfraError -> parameter
                par2InfraErr: parameter -> InfraError
                h_TimeFormat2par (*! constructor *) : h_TimeFormat -> parameter
                par2h_TimeFormat: parameter -> h_TimeFormat
                sec_TimeFormat2par (*! constructor *) : sec_TimeFormat -> parameter
                par2sec_TimeFormat: parameter -> sec_TimeFormat
                titel2par (*! constructor *) : titel -> parameter
                par2titel: parameter -> titel
                tList2par (*! constructor *) : tList -> parameter
                par2tList: parameter -> tList
                ident2par (*! constructor *) : ident -> parameter
                par2ident: parameter -> ident
                identList2par (*! constructor *) : identList -> parameter
                par2identList: parameter -> identList
                Int2par (*! constructor *) : Int -> parameter
                par2Int: parameter -> Int
                e (*! constructor *) : -> plist
                _._ (*! constructor *) : parameter, plist -> plist
                head: plist -> parameter
                tail: plist -> plist
        eqns
         forall p: parameter, id: ident, pl: plist, idL: identList,
                DsE: DsError, CrE: CrError, InE: InfraError,
                hTi: h_TimeFormat, sec: sec_TimeFormat, z: Int,
                t: titel, tL: tList
        ofsort ident
                par2ident( ident2par(id) ) = id;
        ofsort identList
                par2identList( identList2par(idL) ) = idL;
        ofsort parameter
                head( p.pl ) = p;
        ofsort plist
                tail( p.pl ) = pl;
        ofsort CrError
                par2CrErr( CrErr2par(CrE) ) = CrE;
        ofsort DsError
                par2DsErr( DsErr2par(DsE) ) = DsE;
        ofsort InfraError
                par2InfraErr( InfraErr2par(InE) ) = InE;
        ofsort h_TimeFormat
                par2h_TimeFormat( h_TimeFormat2par(hTi) ) = hTi;
        ofsort sec_TimeFormat
                par2sec_TimeFormat( sec_TimeFormat2par(sec) ) = sec;
        ofsort Int
                par2Int( Int2par(z) ) = z;
        ofsort titel
                par2titel( titel2par(t) ) = t;
        ofsort tList
                par2tList( tList2par(tL) ) = tL;
endtype (* parameter_type *)
(* -------------------------------------------------------------------------
        module parameter_type is identifier_type, TimeFormat_Type, Failure_Types, video_titel_type
        type    parameter is
                  CrErr2par (CrE: CrError)
                | DsErr2par (DsE: DsError)
                | InfraErr2par (InE: InfraError)
                | h_TimeFormat2par (hTi: h_TimeFormat)
                | sec_TimeFormat2par (sec: sec_TimeFormat)
                | titel2par (T: titel)
                | tList2par (TL: tList)
                | ident2par (ID: ident)
                | identList2par (IDL: identList)
                | Int2par (I: Int)
        endtype
        type    plist is
                  e
                | _._ (P: parameter, PL: plist)
        endtype

        function par2CrErr(P: parameter) : CrError is select CrE in P endfunc
        function par2DsErr (P: parameter) : DsError is select DsE in P endfunc
        function par2InfraErr (P: parameter) : InfraError is select InE in P endfunc
        function par2h_TimeFormat (P: parameter) : h_TimeFormat is select hTi in P endfunc
        function par2sec_TimeFormat (P: parameter) : sec_TimeFormat is select sec in P endfunc
        function par2titel (P: parameter) : titel is select T in P endfunc
        function par2tList (P: parameter) : tList is select TL in P endfunc
        function par2ident (P: parameter) : ident is select ID in P endfunc
        function par2identList (P: parameter) : identList is select IDL in P endfunc
        function par2Int (P: parameter) : Int select I in P endfunc

        function head (L: plist) : parameter is select P in L endfunc
        function tail (L: plist) : plist is select PL in L endfunc
        endmod
=========================================================================== *)

type role_type is
        sorts RoleType
        opns
                client (*! constructor *), server (*! constructor *),
                stream   (*! constructor *) : -> RoleType
endtype (* role_type *)
(* -------------------------------------------------------------------------
        module role_type is
        type    RoleType is
                  client
                | server
                | stream
        endtype
        endmod
=========================================================================== *)

type internal_type is
        sorts   keyword, template_names, op_term_names, flow_names,
                basic_stream_type_names
        opns
                instantiate (*! constructor *),
                terminate (*! constructor *),
                term_case (*! constructor *),
                invoke (*! constructor *),
                invocation (*! constructor *),
                invoked (*! constructor *),
                obj_creation (*! constructor *),
                stop_if (*! constructor *),
                stop_object (*! constructor *),
                successful (*! constructor *),
                putsignal (*! constructor *),
                getsignal (*! constructor *),
                termination (*! constructor *)  : -> keyword

                binding_object (*! constructor *),
                customer (*! constructor *),
                stereo_video_player (*! constructor *),
                mono_output_signal (*! constructor *),
                stereo_output_signal (*! constructor *),
                mono_input_signal (*! constructor *),
                stereo_input_signal (*! constructor *),
                tv_output_interface (*! constructor *),
                tv_input_interface (*! constructor *),
                video_control_interface (*! constructor *),
                binding_control_interface  (*! constructor *) : -> template_names

                play_movie (*! constructor *),
                connection_established (*! constructor *),
                add_customer (*! constructor *),
                customer_added (*! constructor *),
                remove_customer (*! constructor *),
                connection_terminated (*! constructor *),
                stop_movie (*! constructor *),
                movie_stoped (*! constructor *),
                unbind (*! constructor *),
                binder_object_stoped (*! constructor *),
                add_consumer (*! constructor *),
                consumer_added (*! constructor *),
                close_channel (*! constructor *),
                connection_closed (*! constructor *),
                error (*! constructor *),
                infrastructure_error (*! constructor *) : -> op_term_names

                video_signal (*! constructor *),
                left_channel (*! constructor *),
                right_channel (*! constructor *),
                audio_signal (*! constructor *) : -> flow_names

                AUDIO (*! constructor *),
                VIDEO (*! constructor *) : -> basic_stream_type_names

endtype (* internal_type *)
(* ---------------------------------------------------------------------------
        module internal_type is
        type    keyword is
                  instantiate
                | terminate
                | term_case
                | invoke
                | invocation
                | invoked
                | obj_creation
                | stop_if
                | stop_object
                | successful
                | putsignal
                | getsignal
                | termination
        endtype
        type    template_names is
                  binding_object
                | customer
                | stereo_video_player
                | mono_output_signal
                | stereo_output_signal
                | mono_input_signal
                | stereo_input_signal
                | tv_output_interface
                | tv_input_interface
                | video_control_interface
                | binding_control_interface
        endtype
        type    op_term_names is
                  play_movie
                | connection_established
                | add_customer
                | customer_added
                | remove_customer
                | connection_terminated
                | stop_movie
                | movie_stoped
                | unbind
                | binder_object_stoped
                | add_consumer
                | consumer_added
                | close_channel
                | connection_closed
                | error
                | infrastructure_error
        endtype
        type    flow_names
                  video_signal
                | left_channel
                | right_channel
                | audio_signal
        endtype
        type    basic_stream_type_names is
                  AUDIO
                | VIDEO
        endtype

        endmod (* internal_type *)
============================================================================ *)
```

```
behaviour
(* pas complete *)

hide  interaction, obj_creation, sys_init, produce, consume in

object_space[ interaction, obj_creation, produce, consume ]

|[ interaction,obj_creation,produce,consume ]|

infrastructure[ interaction,obj_creation,sys_init,produce,consume ]

|[ sys_init ]|

( sys_init !obj_creation !stereo_video_player !tList2par( The_piano.(JFK.(Jurassic_park.a )) ).e;
   sys_init !obj_creation !termination !stereo_video_player ?idL1: identList;
    sys_init !obj_creation !customer !ident2par(head(idL1)).(identList2par(o).e);
     sys_init !obj_creation !termination !customer ?idL2: identList;
      sys_init !obj_creation !customer !ident2par(head(idL1)).(identList2par(head(tail(idL2)).o).e);
       sys_init !obj_creation !termination !customer ?idL3: identList;
        stop )

where

process object_space[ interaction, obj_creation, produce, consume ]:
        noexit :=
  obj_creation !stereo_video_player ?p: plist;
  ( stereo_video_player[ interaction,obj_creation,produce,consume ]( par2tList(head(p)) )
   |||
    object_space[ interaction, obj_creation,produce,consume ] )
  []
  obj_creation !binding_object ?p: plist;
  ( binding_object[ interaction,obj_creation,produce,consume ] ( par2identList(head(p)) )
   |||
    object_space[ interaction, obj_creation,produce,consume ] )
  []
  obj_creation !customer ?p: plist;
  ( customer[ interaction,obj_creation,produce,consume ]( par2ident(head(p)), par2identList(head(tail(p))) )
         |||
    object_space[ interaction, obj_creation,produce,consume ] )
where

(* ++++++++++++ BINDING OBJECT TEMPLATE SPECIFICATION START ++++++++++++++++++ *)

process binding_object[ interaction,obj_creation,produce,consume ]( streamIFList: identList )  : noexit :=

  hide ig,stopObj  in
  interface_manager[ interaction,produce,consume,ig,stopObj ]

  |[ ig,stopObj ]|

  binding_object_object_body[ interaction,obj_creation,ig,stopObj  ] ( streamIFList )

where
```

```
process interface_manager[ interaction,produce,consume,ig,stopObj ]: noexit :=
  ( ig !instantiate !binding_control_interface !server ?bind_ctrl_if_id: ident;
    ( SERVER_binding_control_interface[ interaction, ig ] (bind_ctrl_if_id, server)
    |||
     interface_manager[ interaction,produce,consume, ig,stopObj ] )
  []
     ig !instantiate !tv_output_interface !stream ?tv_out_stream_if_id: ident ?conn_end_point: ident;
    ( STREAM_tv_output_interface[ produce,consume, ig ] (tv_out_stream_if_id,conn_end_point)
    |||
     interface_manager[ interaction,produce,consume, ig,stopObj ] )
  []
     ig !instantiate !tv_input_interface !stream ?tv_in_stream_if_id: ident ?conn_end_point: ident;
    ( STREAM_tv_input_interface[ produce,consume, ig ] (tv_in_stream_if_id,conn_end_point)
    |||
     interface_manager[ interaction,produce,consume, ig,stopObj ] )
  )
  [> stopObj !binding_object; STOP

where
   process STREAM_tv_output_interface[ produce,consume, ig ]
    (interface_id: ident, conn_end_point: ident): noexit :=
    FLOW_video_signal[ produce,ig ]( interface_id,conn_end_point )
    |||
    FLOW_audio_signal[ produce,consume,ig ] ( interface_id,conn_end_point )

  where
   process FLOW_video_signal[produce,ig ]
      (interface_id: ident, conn_end_point: ident): noexit :=
    ig !putsignal !video_signal !VIDEO !interface_id !conn_end_point;
     produce !video_signal !VIDEO !interface_id !conn_end_point;
     ( FLOW_video_signal[produce,ig ]( interface_id,conn_end_point )
      []
       produce !video_signal !VIDEO !interface_id !conn_end_point !infrastructure_error;
        FLOW_video_signal[produce,ig ]( interface_id,conn_end_point )
      )
    []
     ig !stop_if !tv_output_interface !stream !interface_id !conn_end_point; STOP
   endproc (* FLOW_video_signal *)

   process FLOW_audio_signal[ produce,consume,ig ]
    ( interface_id: ident, conn_end_point: ident ): noexit :=

    FLOW_left_channel[ produce,ig ]( interface_id,conn_end_point )
    |||
    FLOW_right_channel[ produce,ig ] ( interface_id,conn_end_point )

    where
        process FLOW_left_channel[produce,ig ]
      (interface_id: ident, conn_end_point: ident): noexit :=
    ig !putsignal !left_channel !AUDIO !interface_id !conn_end_point;
     produce !left_channel !AUDIO !interface_id !conn_end_point;
     ( FLOW_left_channel[produce,ig ]( interface_id,conn_end_point )
      []
```

```
        produce !left_channel !AUDIO !interface_id !conn_end_point !infrastructure_error;
         FLOW_left_channel[produce,ig ]( interface_id,conn_end_point )
       )
     []
      ig !stop_if !tv_output_interface !stream !interface_id !conn_end_point; STOP
         endproc (* FLOW_left_channel *)

         process FLOW_right_channel[produce,ig ]
       (interface_id: ident, conn_end_point: ident): noexit :=
     ig !putsignal !right_channel !AUDIO !interface_id !conn_end_point;
      produce !right_channel !AUDIO !interface_id !conn_end_point;
      ( FLOW_right_channel[produce,ig ]( interface_id,conn_end_point )
       []
        produce !right_channel !AUDIO !interface_id !conn_end_point !infrastructure_error;
         FLOW_right_channel[produce,ig ]( interface_id,conn_end_point )
       )
     []
      ig !stop_if !tv_output_interface !stream !interface_id !conn_end_point; STOP
         endproc (* FLOW_right_channel *)

    endproc (* FLOW_audio_signal *)
    endproc (*  STREAM_tv_output_interface / stream interface *)

    process STREAM_tv_input_interface[ produce,consume, ig ]
     (interface_id: ident, conn_end_point: ident): noexit :=
     FLOW_video_signal[ consume,ig ]( interface_id,conn_end_point )
     |||
     FLOW_audio_signal[ produce,consume,ig ]( interface_id,conn_end_point )

   where
    process FLOW_video_signal[consume,ig ]
       (interface_id: ident, conn_end_point: ident): noexit :=
     consume !video_signal !VIDEO !interface_id !conn_end_point;
      ig !getsignal !video_signal !VIDEO !interface_id !conn_end_point;
       FLOW_video_signal[consume,ig ]( interface_id,conn_end_point )
     []
      ig !stop_if !tv_input_interface !stream !interface_id !conn_end_point; STOP
    endproc (* FLOW_video_signal *)

    process FLOW_audio_signal[ produce,consume,ig ]
       ( interface_id: ident, conn_end_point: ident ): noexit :=
     FLOW_left_channel[ consume,ig ]( interface_id,conn_end_point )
     |||
     FLOW_right_channel[ consume,ig ] ( interface_id,conn_end_point )

     where
         process FLOW_left_channel[consume,ig ]
       (interface_id: ident, conn_end_point: ident): noexit :=
     consume !left_channel !AUDIO !interface_id !conn_end_point;
      ig !getsignal !left_channel !AUDIO !interface_id !conn_end_point;
         FLOW_left_channel[consume,ig ]( interface_id,conn_end_point )
         []
      ig !stop_if !tv_input_interface !stream !interface_id !conn_end_point; STOP
         endproc (* FLOW_left_channel *)
         process FLOW_right_channel[consume,ig ]
        (interface_id: ident, conn_end_point: ident): noexit :=
     consume !right_channel !AUDIO !interface_id !conn_end_point;
      ig !getsignal !right_channel !AUDIO !interface_id !conn_end_point;
         FLOW_right_channel[consume,ig ]( interface_id,conn_end_point )
     []
      ig !stop_if !tv_input_interface !stream !interface_id !conn_end_point; STOP
         endproc (* FLOW_right_channel *)

    endproc (* FLOW_audio_signal *)
    endproc (*  STREAM_tv_input_interface / stream interface *)

    process SERVER_binding_control_interface[ interaction,ig ]
      (interface: ident, role: RoleType): noexit :=
    operation_unbind[ interaction,ig ] (interface, role)
    []
    operation_add_consumer[ interaction,ig ] (interface, role )
    []
    operation_close_channel[ interaction,ig ] (interface, role)
    []
    ig !stop_if !binding_control_interface !role !interface; stop

       where
   process operation_unbind[ interaction, ig ]
       ( self: ident, role: RoleType): noexit :=
     interaction !invocation !role ?op_if: ident !self !unbind  ?p: plist;
       ig !invoked !op_if !self !unbind !p;
          ( ig !terminate !op_if !self !binder_object_stoped ?p: plist;
       interaction !termination  !role !op_if !self !binder_object_stoped !p;
          SERVER_binding_control_interface[interaction, ig ](self, role )
          []
       interaction !termination  !role !op_if !self !infrastructure_error ?p: plist;
                SERVER_binding_control_interface[interaction, ig ](self, role ) )
   endproc (* operation_unbind *)

   process operation_add_consumer[interaction, ig ]
       ( self: ident, role: RoleType ): noexit :=
     interaction !invocation !role ?op_if: ident !self !add_consumer ?p: plist;
       ig !invoked !op_if !self !add_consumer !p;
          ( ig !terminate !op_if !self !consumer_added ?p: plist;
           interaction !termination !role !op_if !self !consumer_added !p;
          SERVER_binding_control_interface[ interaction, ig ](self, role )
          []
       ig !terminate !op_if !self !error ?p: plist;
        interaction !termination  !role !op_if !self !error !p;
          SERVER_binding_control_interface[interaction, ig ](self, role )
          []
        interaction !termination  !role !op_if !self !infrastructure_error ?p: plist;
          SERVER_binding_control_interface[ interaction, ig ](self, role ) )
   endproc (* operation_add_consumer *)

   process operation_close_channel[ interaction, ig ]
       ( self: ident, role: RoleType ): noexit :=
     interaction !invocation !role ?op_if: ident !self !close_channel ?p: plist;
       ig !invoked !op_if !self !close_channel !p;
        ( ig !terminate !op_if !self !connection_closed ?p: plist;
           interaction !termination  !role !op_if !self !connection_closed !p;
          SERVER_binding_control_interface[ interaction,ig ](self, role )
          []
       ig !terminate !op_if !self !error ?p: plist;
        interaction !termination  !role !op_if !self !error !p;
          SERVER_binding_control_interface[interaction, ig ](self, role )
          []
               interaction !termination  !role !op_if !self !infrastructure_error ?p: plist;
          SERVER_binding_control_interface[ interaction,ig ](self, role ) )
   endproc (* operation_close_channel *)

    endproc (* SERVER_binding_control_interface / server role *)

 endproc   (*  interface_manager  *)

 process binding_object_object_body[ interaction,obj_creation,ig,stopObj ]
     ( stream_ifaces: identList ): noexit :=
   binding_object_initial_statements[ interaction,obj_creation,ig ](stream_ifaces)
   >>  accept myifList, serverIFList, video_signalConList, left_channelConList,
               right_channelConList: identList   in
     binding_object_internal_behaviour[ interaction,obj_creation,ig,stopObj ]
     ( myifList, serverIFList, stream_ifaces, video_signalConList, left_channelConList,
               right_channelConList )

     where
   process binding_object_initial_statements[ interaction,obj_creation,ig ]
       ( stream_ifaces: identList ): exit(identList,identList,identList,identList,identList) :=
        part1[ interaction,obj_creation,ig ]( stream_ifaces )
      >>  accept myifList, serverIFList : identList in
        loop_process[ interaction,obj_creation,ig ]
     ( myifList, serverIFList, tail(stream_ifaces), o, o, o )
      >>   accept myifList, serverIFList,  video_signalConList, left_channelConList,
               right_channelConList: identList in
       ( obj_creation !successful !binding_object !myifList;
          exit( myifList, serverIFList, video_signalConList, left_channelConList, right_channelConList )   )

   where
     process part1[ interaction,obj_creation,ig ]
     (  stream_ifaces: identList ): exit( identList, identList ) :=
     ig !instantiate !binding_control_interface !server ?binding_ctrl_if_id: ident;
           ig !instantiate !tv_input_interface !stream ?tv_input_if_id: ident !head(stream_ifaces);
       exit( binding_ctrl_if_id.(tv_input_if_id.o) , o )
     endproc (* part1 *)

     process loop_process[ interaction,obj_creation,ig ]
     ( myifList, serverIFList, stream_ifaces, video_signalConList, left_channelConList,
       right_channelConList: identList ): exit( identList, identList, identList, identList, identList ):=
     [ not(empty(stream_ifaces)) = true ] ->  (
        ig !instantiate !tv_output_interface !stream ?tv_output_if_id: ident !head(stream_ifaces);
         loop_process[ interaction,obj_creation,ig ]
     ( tv_output_if_id.myifList, serverIFList, tail(stream_ifaces),tv_output_if_id.video_signalConList,
        tv_output_if_id.left_channelConList, tv_output_if_id.right_channelConList  )
                )
      []
      [ empty(stream_ifaces) = true ] ->  (
          exit( myifList, serverIFList, video_signalConList, left_channelConList, right_channelConList  )
                )
      endproc (* loop_process *)

    endproc (* binding_object_initial_statements *)

    process binding_object_internal_behaviour[interaction,obj_creation,ig,stopObj ]
      ( myifList, serverIFList, stream_ifaces, video_signalConList, left_channelConList,
        right_channelConList: identList ): noexit :=
(*      ( myifList: identList, serverIFList: identList, stream_ifaces: identList ) : noexit :=    *)

      ig !invoked ?client_if: ident ?binding_cntrl_if: ident !unbind ?p: plist;
          ig !terminate !client_if !binding_cntrl_if !binder_object_stoped !e;
      stopObj !binding_object;
       STOP

      []
      ig !invoked ?client_if: ident ?binding_cntrl_if: ident !add_consumer ?p: plist;
        ig !instantiate !tv_output_interface !stream ?tv_output_if_id: ident !par2ident(head(p));
          ig !terminate !client_if !binding_cntrl_if !consumer_added !e;
       binding_object_internal_behaviour[interaction,obj_creation,ig,stopObj ]
      ( tv_output_if_id.myifList, serverIFList, par2ident(head(p)).stream_ifaces,
        tv_output_if_id.video_signalConList, tv_output_if_id.left_channelConList,
        tv_output_if_id.right_channelConList )

      []
      ig !invoked ?client_if: ident ?binding_cntrl_if: ident !close_channel ?p: plist;
       ( [ inlist(par2ident(head(p)),stream_ifaces) = true ] ->
        ( ig !stop_if !tv_output_interface !stream ?if_id: ident !par2ident(head(p));
          ig !terminate !client_if !binding_cntrl_if !connection_closed !e;
          binding_object_internal_behaviour[interaction,obj_creation,ig,stopObj ]
          ( remove(if_id,myifList), serverIFList, remove( par2ident(head(p)),stream_ifaces ),
            remove(if_id,video_signalConList), remove(if_id,left_channelConList),
            remove(if_id,right_channelConList)
          )
         )
          []
          [ not(inlist(par2ident(head(p)),stream_ifaces)) = true ] ->
         ( ig !terminate !client_if !binding_cntrl_if !error !DsErr2par(no_such_interface).e;
           binding_object_internal_behaviour[interaction,obj_creation,ig,stopObj ]
           ( myifList, serverIFList, stream_ifaces,video_signalConList, left_channelConList,
             right_channelConList )
          )
       )

(*   INFORMATION FLOW HANDLING & ROUTING WITHIN THE BINDING-OBJECT :     *)

      []
      ig !getsignal !video_signal !VIDEO ?tv_input_if_id: ident ?conn_end_point: ident;
       ( FLOW_ROUTING_video_signal[ ig ]( video_signalConList )
        |||
        binding_object_internal_behaviour[interaction,obj_creation,ig,stopObj ]
        ( myifList, serverIFList, stream_ifaces, video_signalConList, left_channelConList,
               right_channelConList ) )

     []
     ig !getsignal !left_channel !AUDIO ?tv_input_if_id: ident ?conn_end_point: ident;
      ( FLOW_ROUTING_left_channel[ ig ]( left_channelConList )
        |||
        binding_object_internal_behaviour[interaction,obj_creation,ig,stopObj ]
         ( myifList, serverIFList, stream_ifaces,video_signalConList, left_channelConList,
               right_channelConList ) )

     []
     ig !getsignal !right_channel !AUDIO ?tv_input_if_id: ident ?conn_end_point: ident;
      ( FLOW_ROUTING_right_channel[ ig ]( right_channelConList )
        |||
        binding_object_internal_behaviour[interaction,obj_creation,ig,stopObj ]
        ( myifList, serverIFList, stream_ifaces,video_signalConList, left_channelConList,
               right_channelConList ) )

   where

      process FLOW_ROUTING_video_signal[ ig ]( ConnList: identList ):noexit :=
     [ not(empty(ConnList)) = true ] ->
         ( ig !putsignal !video_signal !VIDEO !head(ConnList) ?conn_end_point: ident;
            FLOW_ROUTING_video_signal[ ig ] ( tail(ConnList) )
         )
     []
     [ empty(ConnList) = true ] -> ( STOP )
      endproc (* FLOW_ROUTING_video_signal *)

      process FLOW_ROUTING_left_channel[ ig ]( ConnList: identList ):noexit :=
     [ not(empty(ConnList)) = true ] ->
         ( ig !putsignal !right_channel !AUDIO !head(ConnList) ?conn_end_point: ident;
            FLOW_ROUTING_left_channel[ ig ] ( tail(ConnList) )
         )
     []
     [ empty(ConnList) = true ] -> ( STOP )
      endproc (* FLOW_ROUTING_left_channel *)

      process FLOW_ROUTING_right_channel[ ig ]( ConnList: identList ):noexit :=
     [ not(empty(ConnList)) = true ] ->
         ( ig !putsignal !left_channel !AUDIO !head(ConnList) ?conn_end_point: ident;
            FLOW_ROUTING_right_channel[ ig ] ( tail(ConnList) )
         )
     []
     [ empty(ConnList) = true ] -> ( STOP )
      endproc (* FLOW_ROUTING_right_channel *)

   endproc (* binding_object_internal_behaviour *)

 endproc (* binding_object_object_body *)
```

```
endproc (* object template binding_object *)


(* ++++++++++++++ BINDING-OBJECT TEMPLATE END ++++++++++++++++++++++++++++++++++++ *)

(* ++++++++++++++ STEREO-VIDEO-PLAYER OBJECT-TEMPLATE START ++++++++++++++++++++++ *)

process stereo_video_player[ interaction,obj_creation,produce,consume ]( video_storage: tList )  :
        noexit :=
  hide ig,stopObj  in
  interface_manager[ interaction,produce,consume,ig,stopObj ]
  |[ ig,stopObj ]|
  stereo_video_player_object_body[ interaction,obj_creation,ig,stopObj  ] ( video_storage )

where
  process interface_manager[ interaction,produce,consume,ig,stopObj ]: noexit :=
  (   ig !instantiate !video_control_interface !server ?video_if_id: ident;
      ( SERVER_video_control_interface[ interaction, ig ] (video_if_id, server)
      |||
       interface_manager[ interaction,produce,consume, ig,stopObj ] )
          []
     ig !instantiate !binding_control_interface !client ?bind_ctrl_if_id: ident;
      ( CLIENT_binding_control_interface[ interaction, ig ] (bind_ctrl_if_id, client)
      |||
       interface_manager[ interaction,produce,consume, ig,stopObj ] )
    []
       ig !instantiate !tv_output_interface !stream ?stream_if_id: ident;
      ( STREAM_tv_output_interface[ produce,consume, ig ] (stream_if_id)
      |||
       interface_manager[ interaction,produce,consume, ig,stopObj ] )
  )
  [> stopObj !stereo_video_player; STOP

  where
     process STREAM_tv_output_interface[ produce,consume, ig ]
      (interface_id: ident): noexit :=

      FLOW_video_signal[ produce,ig ]( interface_id )
      |||
      FLOW_audio_signal[ produce,consume,ig ] ( interface_id )

    where
     process FLOW_video_signal[produce,ig ](interface_id: ident): noexit :=
      produce !video_signal !VIDEO !interface_id;
       ( FLOW_video_signal[produce,ig ]( interface_id )
        []
         produce !video_signal !VIDEO !interface_id !infrastructure_error;
          FLOW_video_signal[produce,ig ]( interface_id )
        )
      []
       ig !stop_if !tv_output_interface !stream !interface_id; STOP
     endproc (* FLOW_video_signal *)

     process FLOW_audio_signal[ produce,consume,ig ] ( interface_id: ident ): noexit :=
      FLOW_left_channel[ produce,ig ]( interface_id )
      |||
      FLOW_right_channel[ produce,ig ] ( interface_id )

      where
        process FLOW_left_channel[produce,ig ](interface_id: ident): noexit :=
      produce !left_channel !AUDIO !interface_id;
       ( FLOW_left_channel[produce,ig ]( interface_id )
         []
         produce !left_channel !AUDIO !interface_id !infrastructure_error;
          FLOW_left_channel[produce,ig ]( interface_id )
         )
       []
       ig !stop_if !tv_output_interface !stream !interface_id; STOP
          endproc (* FLOW_left_channel *)

          process FLOW_right_channel[produce,ig ](interface_id: ident): noexit :=
      produce !right_channel !AUDIO !interface_id;
       ( FLOW_right_channel[produce,ig ]( interface_id )
         []
         produce !right_channel !AUDIO !interface_id !infrastructure_error;
          FLOW_right_channel[produce,ig ]( interface_id )
         )
       []
       ig !stop_if !tv_output_interface !stream !interface_id; STOP
          endproc (* FLOW_right_channel *)

     endproc (* FLOW_audio_signal *)
     endproc (*  STREAM_tv_output_interface / stream interface *)

     process SERVER_video_control_interface[ interaction,ig ]
      (interface: ident, role: RoleType): noexit :=
     operation_play_movie[ interaction,ig ] (interface, role)
     []
     operation_add_customer[ interaction,ig ] (interface, role )
     []
     operation_remove_customer[ interaction,ig ] (interface, role)
     []
     operation_stop_movie[ interaction,ig ] (interface, role)
     []
     ig !stop_if !video_control_interface !role !interface; stop

        where

    process operation_play_movie[ interaction, ig ]
      ( self: ident, role: RoleType): noexit :=
      interaction !invocation !role ?op_if: ident !self !play_movie  ?p: plist;
        ig !invoked !op_if !self !play_movie !p;
         ( ig !terminate !op_if !self !connection_established ?p: plist;
           interaction !termination  !role !op_if !self !connection_established !p;
           SERVER_video_control_interface[interaction, ig ](self, role )
            []
        ig !terminate !op_if !self !error ?p: plist;
         interaction !termination  !role !op_if !self !error !p;
           SERVER_video_control_interface[interaction, ig ](self, role )
           []
        interaction !termination  !role !op_if !self !infrastructure_error ?p: plist;
                 SERVER_video_control_interface[interaction, ig ](self, role ) )
    endproc (* operation_create_stopwatch *)

    process operation_add_customer[interaction, ig ]
        ( self: ident, role: RoleType ): noexit :=
      interaction !invocation !role ?op_if: ident !self !add_customer ?p: plist;
        ig !invoked !op_if !self !add_customer !p;
           ( ig !terminate !op_if !self !customer_added ?p: plist;
            interaction !termination  !role !op_if !self !customer_added !p;
           SERVER_video_control_interface[ interaction, ig ](self, role )
           []
        ig !terminate !op_if !self !error ?p: plist;
         interaction !termination  !role !op_if !self !error !p;
           SERVER_video_control_interface[interaction, ig ](self, role )
           []
         interaction !termination  !role !op_if !self !infrastructure_error ?p: plist;
           SERVER_video_control_interface[ interaction, ig ](self, role ) )
    endproc (* operation_destroy_clock *)

    process operation_remove_customer[ interaction, ig ]
        ( self: ident, role: RoleType ): noexit :=
      interaction !invoc
```

Chapter 6

LAPB Protocol

6.1 Comments

From our analysis of the formal description of LAPB protocol, we can make the following remarks:

- There are a lot of enumerated types. For these types (predefined) tester functions are useful.
- The actualization and renaming are unified in a single step (like in IS standard). See for this types like `Pacqueue` and `Framequeue`.
- If the built-in natural type is introduced, the module `DecimalConstants` will become useless.
- If syntactical equality is sutomatically generated, the function `h` will become useless. For example, in the type `Frametype`, the `h` function is used only for equality and tester functions definition; it maps the constructors of sort `frame` onto natural numbers. The LOTOS definition is:

```
make_iframe (*! constructor *)
        : flag, adress, ctl, packet, fcs, flag  -> frame
make_rr     (*! constructor *)
        : flag, adress, ctl,fcs, flag  -> frame
make_rnr    (*! constructor *)
        : flag, adress, ctl, fcs, flag -> frame
make_rej    (*! constructor *)
        : flag, adress, ctl, fcs, flag -> frame
make_sabm   (*! constructor *)
        : flag, adress, ctl, fcs, flag -> frame
make_sabme  (*! constructor *)
        : flag, adress, ctl, fcs, flag -> frame
make_disc   (*! constructor *)
        : flag, adress, ctl, fcs, flag -> frame
make_ua     (*! constructor *)
        : flag, adress, ctl, fcs, flag -> frame
make_dm     (*! constructor *)
        : flag, adress, ctl, fcs, flag -> frame
make_frmr   (*! constructor *)
        : flag, adress, ctl, frmrreason, fcs, flag  -> frame
h           : frame  ->  Nat
```

```
      ofsort Nat
        h(make_iframe(f1,a, c, p, fcc, f2)) = 0;
        h(make_rr(f1, a, c, fcc, f2))       = Succ(0) ;
        h(make_rnr(f1, a, c, fcc, f2))      = Succ(Succ(0));
        h(make_rej(f1, a, c, fcc, f2))      = Succ(Succ(Succ(0)));
        h(make_sabm(f1, a, c, fcc, f2))     = Succ(Succ(Succ(Succ(0))));
        h(make_sabme(f1, a, c, fcc, f2))    = Succ(h(make_sabm(f1, a, c, fcc, f2))) ;
        h(make_disc(f1, a, c, fcc, f2))     = Succ(h(make_sabme(f1, a, c, fcc, f2)));
        h(make_ua(f1, a, c, fcc, f2))       = Succ(h(make_disc(f1, a, c, fcc, f2)));
        h(make_dm(f1, a, c, fcc, f2))       = Succ(h(make_ua(f1, a, c, fcc, f2)));
        h(make_frmr(f1, a, c, inf, fcc, f2))     = Succ(h(make_dm(f1, a, c, fcc, f2)));

      ofsort Bool
        is_i(f) = (h(f) eq 0);
```

A direct translation in E-LOTOS is:

```
      type  frame is
          make_iframe (F1: flag, A: adress, C: ctl, P: packet, FCC: fcs, F2: flag)
        | make_rr     (F1: flag, A: adress, C: ctl, FCC: fcs, F2: flag)
        | make_rnr    (F1: flag, A: adress, C: ctl, FCC: fcs, F2: flag)
        | make_rej    (F1: flag, A: adress, C: ctl, FCC: fcs, F2: flag)
        | make_sabm   (F1: flag, A: adress, C: ctl, FCC: fcs, F2: flag)
        | make_sabme  (F1: flag, A: adress, C: ctl, FCC: fcs, F2: flag)
        | make_disc   (F1: flag, A: adress, C: ctl, FCC: fcs, F2: flag)
        | make_ua     (F1: flag, A: adress, C: ctl, FCC: fcs, F2: flag)
        | make_dm     (F1: flag, A: adress, C: ctl, FCC: fcs, F2: flag)
        | make_frmr   (F1: flag, A: adress, C: ctl, INF: frmrreason, FCC: fcs, F2: flag)
      endtype
    function h (F: frame) : Nat is
        case F in
        | make_iframe (...) -> 0
        | make_rr (...) -> 1
        | make_rnr (...) -> 2
        | make_rej (...) -> 3
        | make_sabm (...) -> 4
        | make_sabme (...) -> 5
        | make_disc (...) -> 6
        | make_ua (...) -> 7
        | make_dm (...) -> 8
        | make_frmr (...) -> 9
        endcase
     endfunc
     function is_i (f: frame) : Bool is (h(f) eq 0) endfunc
```

However, by using a (predefined) tester function the `h` function becomes unnecessary, and the definition of `is_i` function can be rewritten as:

```
     function is_i (f: frame) : Bool is f match make_iframe (...) endfunc
```

Also, the `is_s` function can be defined in two forms:

```
    function is_s  (f: frame) : Bool is
        (is_rr(f) or is_rnr(f)) or (is_rej(f))
     endfunc
```

```
function is_s  (f: frame) : Bool is
(f match make_rnr (...)) or
(f match make_rr (...)) or
(f match make_rej (...))
endfunc
```

We mention that the `h` function is not used in the behaviour part of the specification.

- The same treatment applies for the **`eval`** function. The LOTOS definition is:

```
  eval          : primitive -> Nat
  is_dl_est_req : primitive -> Bool
  is_dl_dis_req : primitive -> Bool
  ...

ofsort Nat
  eval(dl_est_req) = 0;
  eval(dl_dis_req) = Succ(0);
  ...
ofsort Bool
  is_dl_est_req(p) = (eval(p) eq 0);
  is_dl_dis_req(p) = (eval(p) eq Succ(0));
  ...
```

Direct translation in E-LOTOS is:

```
function eval (P: primitive) : Nat is
    case P in
        dl_est_req -> 0
      | dl_dis_req -> 1
      | dl_dis_cnf -> 2
      | dl_est_ind -> 3
      | dl_est_cnf -> 4
      | dl_dis_ind -> 5
      | dl_data_req (...) -> 6
      | dl_data_ind (...) -> 7
    endcase
endfunc
function is_dl_est_req (P: primitive) : Bool is eval(P) eq 0  endfunc
function is_dl_dis_req (P: primitive) : Bool is eval(P) eq 1  endfunc
...
```

However, by using a (predefined) tester function the **`eval`** function becomes unnecessary, and tester functions could be rewritten as:

```
function is_dl_est_req (P: primitive) : Bool is P match dl_est_req endfunc
function is_dl_dis_req (P: primitive) : Bool is P match dl_dis_req endfunc
...
```

In the control part of specification, the use of **`eval`** function could be avoided by pattern-matching.

- In the specification of type `Constraint`, the equations defining `I_Sxx` functions contain a free variable `p` in the right part of the equations. We translated this in **`any packet`**.

- The k function is used in a guard which is commented `(* [k lt k(tmode)] -> *)` (lapb process).

- However, there are cases where the syntactical equality is not used. In fact, the equality of `eframe` sort is defined like the (syntactical) equality of `frame` components. In such cases, definition of equality could be overwritten.

  For example, the (LOTOS) definition of equality for values of `eframe` sort is:

```
  _eq_      : eframe,eframe -> Bool
 f eq f1 = h(get_f(f)) eq h(get_f(f1)) ;
```

  which can be translated in the E-LOTOS function:

```
   function _eq_ (ef1: eframe, ef2: eframe) : Bool is
           get_f(ef1) eq get_f(ef2)
   endfunc
```

- The pattern matching can be used in several places of the control part of the specification.

  1. In the definition of the `infphase` process we can use pattern matching. The LOTOS specification is:

```
(
  [typ eq 0 of Nat] ->
   (
    infphase[dl, phl, p, t](notwait, reject, vsp, init, from,
          to, n2, k, tmode, vs, vr, lu, q, qdls, starttime,
          rbusy, lbusy, chkpt, qdlr, retrans)
   )
  []
  [typ eq Succ(0) of Nat] ->
    (
     hide t in
             t ! start;
              infphase[dl, phl, p, t](notwait, reject, vsp, init,
                from, to, n2, k, tmode, vs, vr, lu, q, qdls,
                starttime, rbusy, lbusy, chkpt, qdlr, retrans)
   )
  []
  [typ eq Succ(Succ(0)) of Nat] ->
     t ! start;
       infphase[dl, phl, p, t](notwait, reject, vsp, init, from,
         to, n2, k, tmode, vs, vr, lu, q, qdls, starttime,
         rbusy, lbusy, chkpt, qdlr, retrans)
  []
  [typ eq Succ(Succ(Succ(0))) of Nat] ->
    infphase[dl, phl, p, t](notwait, reject, vsp, init, from,
      to, n2, k, tmode, vs, vr, lu, q, qdls, starttime,
      rbusy, lbusy, chkpt, qdlr, retrans)
  []
  [typ eq Succ(Succ(Succ(Succ(0)))) of Nat] ->
   (
   p ! lreset ! ind ! rs ! empty of pqueue ! qdls ! qdlr !
   b !  vs ! vr ! lu;
```

```
      stop
     )
)
```

The translation in (E-LOTOS) is:

```
(case typ in
  0 -> ( infphase[dl, phl, p, t](notwait, reject, vsp, init, from,
                to, n2, k, tmode, vs, vr, lu, q, qdls, starttime,
                rbusy, lbusy, chkpt, qdlr, retrans)
               )
| 1 -> (  hide t in
                t ! start;
                infphase[dl, phl, p, t](notwait, reject, vsp, init,
                from, to, n2, k, tmode, vs, vr, lu, q, qdls,
                starttime, rbusy, lbusy, chkpt, qdlr, retrans)
               )
| 2 -> t ! start;
               infphase[dl, phl, p, t](notwait, reject, vsp, init, from,
               to, n2, k, tmode, vs, vr, lu, q, qdls, starttime,
               rbusy, lbusy, chkpt, qdlr, retrans)
| 3 -> infphase[dl, phl, p, t](notwait, reject, vsp, init, from,
               to, n2, k, tmode, vs, vr, lu, q, qdls, starttime,
               rbusy, lbusy, chkpt, qdlr, retrans)
| 4 -> ( p ! lreset ! ind ! rs ! empty of pqueue ! qdls ! qdlr !
               b !  vs ! vr ! lu;
               stop
        )
)
```

2. In the definition of the `sendi` process we can use the “case” or “if-then” statements. For example, the LOTOS specification:

```
 ([not(starttime)] ->
    (t! start
     ;exit(((vs + Succ(0) of Nat) mod k), vr, lu, vsp, (firstone(qdls) +-- q),
     removefirst(qdls) of pqueue, qdlr, true of Bool, rbusy, lbusy, chkpt,
     retrans, reject, init, notwait, 0 of Nat, nors, 0 of Nat, 0 of Bit)
       )
[]
 [starttime] ->
    exit(((vs + Succ(0) of Nat) mod k), vr, lu, vsp, (firstone(qdls) +-- q),
    removefirst(qdls) of pqueue, qdlr, true of Bool, rbusy, lbusy, chkpt,
    retrans, reject, init, notwait, 0 of Nat, nors, 0 of Nat, 0 of Bit)
 )
```

could be translated in the following (E-LOTOS) specification:

```
 (if starttime then
      (t! start
      ;exit(((vs + Succ(0) of Nat) mod k), vr, lu, vsp, (firstone(qdls) +-- q),
      removefirst(qdls) of pqueue, qdlr, true of Bool, rbusy, lbusy, chkpt,
      retrans, reject, init, notwait, 0 of Nat, nors, 0 of Nat, 0 of Bit)
       )
```

```
    else
        exit(((vs + Succ(0) of Nat) mod k), vr, lu, vsp, (firstone(qdls) +-- q),
        removefirst(qdls) of pqueue, qdlr, true of Bool, rbusy, lbusy, chkpt,
        retrans, reject, init, notwait, 0 of Nat, nors, 0 of Nat, 0 of Bit)
    )
```

3. The same transformations apply for processes `releasequeue` and `receiverr`

- Function `flg1` could be rewritten in the following ways:

  1. using a "**let**" statement:

     ```
     function flg1 (F: eframe) : flag is
         let m(FR:=FR: frame,...) = F in get_flag1(FR)
     endfunc
     ```

  2. using a "**select**" statement:

     ```
     function flg1 (F: eframe) : flag is
         get_flag1(select FR in F)
     endfunc
     ```

  3. if the implementation of the `eframe` type is not hidden, we can write the `get_flag1` function call as:

     ```
     function flg1 (F: eframe) : flag is
         select F1 in select FR in F
     endfunc
     ```

- Functions like `is_inforr` and `corrl` could be specified in the following ways:

  1. using other functions:

     ```
     function corrl (F: eframe) : Bool is is_correct(get_c(F)) endfunc
     ```

  2. using a "**select**" statement if the constructors are not hidden:

     ```
     function corrl (F: eframe) : Bool is is_correct(select C in F) endfunc
     ```

  3. using tester and selector statements:

     ```
     function corrl (F: eframe) : Bool is (select C in F) match correct endfunc
     ```

  In consequence, if implementation of types is not hidden, a lot of (user-defined) selector and tester functions are useless.

- The function `makecorruptadr` could be specified in the following ways:

  1. translating conditional equations in guarded "**case**" statements:

```
case true in
  true when is_i(get_f(F)) ->  m(make_iframe(flg1(F),
             corrupt_adr(adrs(F)), ctrl(F), pack(F), fcss(F),
             flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
| true when is_rr(get_f(F)) ->  m(make_rr(flg1(F),
             corrupt_adr(adrs(F)), ctrl(F), fcss(F),
             flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
| true when is_rnr (...) -> m(make_rnr(flg1(F),
             corrupt_adr(adrs(F)), ctrl(F), fcss(F),
             flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
| true when is_rej (...) -> m(make_rej(flg1(F),
             corrupt_adr(adrs(F)), ctrl(F), fcss(F),
             flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
| true when is_sabm (...) -> m(make_sabm(flg1(F),
             corrupt_adr(adrs(F)), ctrl(F), fcss(F),
             flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
| true when is_sabme (...) -> m(make_sabme(flg1(F),
             corrupt_adr(adrs(F)), ctrl(F), fcss(F),
             flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
| true when is_disc (...) -> m(make_disc(flg1(F),
             corrupt_adr(adrs(F)), ctrl(F), fcss(F),
             flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
| true when is_ua (...) -> m(make_ua(flg1(F),
             corrupt_adr(adrs(F)), ctrl(F), fcss(F),
             flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
| true when is_dm (...) -> m(make_dm(flg1(F),
             corrupt_adr(adrs(F)), ctrl(F), fcss(F),
             flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
| true when is_frmr(...) -> m(make_frmr(flg1(F),
             corrupt_adr(adrs(F)), ctrl(F), reasfrmr(F), fcss(F),
             flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
endcase
```

2. using pattern matching in a "**case**" statement:

```
function makecorruptadr (F: eframe) : eframe is
    case (select FR in F) in
      make_iframe (...) -> m(make_iframe(flg1(F),
            corrupt_adr(adrs(F)), ctrl(F), pack(F), fcss(F),
            flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
    | make_rr (...) -> m(make_rr(flg1(F),
            corrupt_adr(adrs(F)), ctrl(F), fcss(F),
            flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
    | make_rnr (...) -> m(make_rnr(flg1(F),
            corrupt_adr(adrs(F)), ctrl(F), fcss(F),
            flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
    | make_rej (...) -> m(make_rej(flg1(F),
            corrupt_adr(adrs(F)), ctrl(F), fcss(F),
            flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
    | make_sabm (...) -> m(make_sabm(flg1(F),
            corrupt_adr(adrs(F)), ctrl(F), fcss(F),
            flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
    | make_sabme (...) -> m(make_sabme(flg1(F),
            corrupt_adr(adrs(F)), ctrl(F), fcss(F),
                  flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
            | make_disc (...) -> m(make_disc(flg1(F),
                  corrupt_adr(adrs(F)), ctrl(F), fcss(F),
                  flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
            | make_ua (...) -> m(make_ua(flg1(F),
                  corrupt_adr(adrs(F)), ctrl(F), fcss(F),
                  flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
            | make_dm (...) -> m(make_dm(flg1(F),
                  corrupt_adr(adrs(F)), ctrl(F), fcss(F),
                  flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
            | make_frmr(...) -> m(make_frmr(flg1(F),
                  corrupt_adr(adrs(F)), ctrl(F), reasfrmr(F), fcss(F),
                  flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
        endcase
    endfunc
```

3. usign updaters (for "sum" and "union" types):

```
function makecorruptadr (F: eframe) : eframe is
F.{FR:= (select FR in F).{A:= corrupt_addr(select A in (select FR in F))}}
endfunc
```

- The same translations apply for the functions `makeundefctl` and `makecorruptfc`.
- Functions like `makenotcorrlength` and `makelesslength` could be specified in two ways:

1. without using updaters:

```
function makenotcorrlength (F: eframe) : eframe is
            m(get_f(F), get_infor(F), incorrect,get_l(F), get_a(F))
endfunc
```

2. using updaters:

```
function makenotcorrlength (F: eframe) : eframe is
        F.{C:=incorrect}
endfunc
```

- The function `makeinfor` could be specified in two ways:

1. using an "**if**" statement:

```
function makeinfor  (F: eframe) : eframe is
    if is_i(get_f(F)) then F
    else m(get_f(F), infor, get_c(F), get_l(F), get_a(F))
endfunc
```

2. using a "**case**" statement:

```
function makeinfor  (F: eframe) : eframe is
    case select FR in F in
      make_iframe (...) -> F
      | any frame -> m(get_f(F), infor, get_c(F), get_l(F), get_a(F))
    endcase
```

```
          endfunc
```

- An example of a "true" pattern-matching "**let**" statement is the specification of functions `w`, `x`, `y`, `z` in the type `wxyz`:

  ```
  function w (BS: BitString) : Bit is
    let (W:Bit) + ((X:Bit) + ((Y:Bit) + (Z:Bit)) = BS in W
  endfunc
  ```

6.2 Formal description in both LOTOS and E-LOTOS

```
 specification lapbl[L, TM](n2, resolvecollis: Nat, k: Nat,
        senddmoption: Bool, tmode: mmode) : noexit
 library
           NaturalNumber, Boolean, BitString, Bit, DecNatRepr,
           DecString, DecDigit
(*
           NaturalNumber, Boolean, BitString, Bit, DecNatRepr,
           String, Element, FBoolean
*)
 endlib

 type Time is
      sorts time
      opns
        start   (*! constructor *) : -> time
        expired (*! constructor *) : -> time
        reset   (*! constructor *) : -> time
 endtype
(* ------------------------------------------------------------------------
 module Time is
      type time is
          start
        | expired
        | reset
      endtype
 endmod
========================================================================== *)

 type Termine is
      sorts term
      opns
        termine      (*! constructor *) : -> term
        causetermine (*! constructor *) : -> term
 endtype
(* ------------------------------------------------------------------------
 module Termine is
      type term is
          termine
        | causetermine
      endtype
 endmod
```

```
========================================================================== *)

 type Linkreset is
      sorts linkreset
      opns
        causelreset (*! constructor *) : -> linkreset
        lreset      (*! constructor *) : -> linkreset
 endtype
(* ------------------------------------------------------------------------
 module Linkreset is
      type linkreset is
          causelreset
        | lreset
      endtype
 endmod
========================================================================== *)

 type Packet is BitString
      renamedby
      sortnames packet for BitString
 endtype
(* ------------------------------------------------------------------------
 module Packet is BitString{sorts packet for BitString}
 endmod
========================================================================== *)

 type Mode is NaturalNumber,Boolean
      sorts mmode
      opns
        basic    (*! constructor *) :  ->  mmode
        extended (*! constructor *) :  ->  mmode
        k        : mmode  ->  Nat
        eq       : mmode, mmode -> Bool
      eqns
      ofsort Nat
        k(basic)    = Succ(Succ(0)) ** (Succ(Succ(Succ(0)))) ;
        k(extended) = Succ(Succ(0)) ** (Succ(Succ(Succ(Succ(Succ(Succ
                      (Succ(0))))))) ;
      ofsort Bool
        eq(basic,extended)  = false;
        eq(basic,basic)     = true of Bool;
        eq(extended,basic)  = false;
        eq(extended,extended) = true of Bool;
 endtype
(* ------------------------------------------------------------------------
 module Mode is
 import NaturalNumber + Boolean
      type mmode
          basic
        | extended
      endtype

      function k (MM: mmode) : Nat is
        case MM in
```

```
            basic -> 2 ** 3
          | extended -> 2 ** 7
        endcase
      endfunc
 endmod
============================================================================ *)

 type Role is Adress, Boolean
      sorts role
      opns
        dte (*! constructor *) :  -> role
        dce (*! constructor *) :  -> role
        adr : role -> adress
        eq  : role,role -> Bool
      eqns
      ofsort adress
        adr(dte) = a;
        adr(dce) = b;
      ofsort Bool
        eq(dte,dte) = true of Bool;
        eq(dte,dce) = false;
        eq(dce,dce) = true of Bool;
        eq(dce,dte) = false;
 endtype
(* ---------------------------------------------------------------------------
 module Role is Adress, Boolean
      type role is
          dte
        | dce
      endtype

      function adr (R: role) : adress is
        case R in
              dte -> a
            | dcs -> b
        endcase
      endfunc
 endmod
============================================================================ *)

 type Adress is Boolean
      sorts adress
      opns
        a               (*! constructor *) :  ->  adress
        b               (*! constructor *) :  ->  adress
        is_a            : adress  ->  Bool
        is_b            : adress  ->  Bool
        corrupt_adr     (*! constructor *) : adress  ->  adress
        is_corrupt_adr  : adress  ->  Bool
        eq              : adress,adress -> Bool
      eqns
      forall x : adress
      ofsort Bool
        is_a(a) = true of Bool ;
```

```
       is_a(b) = false ;
       is_b(a) = false ;
       is_b(b) = true of Bool ;
       is_corrupt_adr(a) = false ;
       is_corrupt_adr(b) = false ;
       is_corrupt_adr(corrupt_adr(x)) = true of Bool ;
       eq(a,a)  = true of Bool;
       eq(a,b)  = false;
       eq(b,a) = false;
       eq(b,b)  = true of Bool;
 endtype
(* --------------------------------------------------------------------------
 module Adress is Boolean
      type adress is
          a
        | b
        | corrupt_adr (adress)
      endtype
      fuction is_a (A: adress) : Bool is A match a endfunc

      function is_b (A: adress) : Bool is A match b endfunc

      function is_corrupt_adr (A: adress) : Bool is A match corrupt_adr(...) endfunc
 endmod
========================================================================= *)

 type Flag is Boolean
      sorts flag
      opns
        flagging  (*! constructor *) :       ->  flag
        noflag    (*! constructor *) : flag  ->  flag
        is_flag   : flag  ->  Bool
      eqns
      forall f: flag
      ofsort Bool
        is_flag(flagging)  =  true of Bool;
        is_flag(noflag(f)) =  false;
 endtype
(* --------------------------------------------------------------------------
 module Flag is Boolean
      type flag is
          flagging
        | noflag (flag)
      endtype

      function is_flag (F: flag) : Bool is F match flagging endfunc
 endmod
========================================================================= *)

 type Infor is Boolean
    sorts inforr
    opns
      infor     (*! constructor *) : -> inforr
      noinfor   (*! constructor *) : -> inforr
```

```
     is_infor : inforr -> Bool
   eqns
   ofsort Bool
     is_infor(infor) = true of Bool;
     is_infor(noinfor) = false;
 endtype
(* ------------------------------------------------------------------------
 module Infor is Boolean
   type inforr is
       infor
     | noinfor
   endtype

   function is_infor (I: inforr) : Bool is I match infor endfunc
 endmod
======================================================================== *)

 type Corrlength is Boolean
   (* type greater for max established for i frame and super unember frame *)
     sorts corrlength
     opns
       correct (*! constructor *) :  ->  corrlength
       incorrect (*! constructor *) : -> corrlength
       is_correct : corrlength -> Bool
     eqns
     ofsort Bool
       is_correct(correct) = true of Bool;
       is_correct(incorrect) = false;
 endtype
(* ------------------------------------------------------------------------
 module Corrlength is Boolean
     type corrlength is
         correct
       | incorrect
     endtype
     function is_correct (CL: corrlength) : Bool is CL match correct endfunc
 endmod
======================================================================== *)

 type Lesslength is Boolean
     sorts lesslength
     opns
       less (*! constructor *) :  -> lesslength
       notless (*! constructor *) :  -> lesslength
       is_not_less  : lesslength  -> Bool
     eqns
     ofsort Bool
       is_not_less(less) = false;
       is_not_less(notless) = true of Bool;
 endtype
(* ------------------------------------------------------------------------
 module Lesslength is Boolean
     type lesslength is
         less
```

```
        | notless
      endtype

      function is_not_less  (LL: lesslength) : Bool is LL match notless endfunc
 endmod
========================================================================== *)

 type Abortf is Boolean
      sorts abortf
      opns
        abort    (*! constructor *) :          ->  abortf
        noabort  (*! constructor *) :          ->  abortf
        is_abort :  abortf  ->  Bool
      eqns
      ofsort Bool
        is_abort(abort)   = true of Bool;
        is_abort(noabort) = false;
 endtype
(* -------------------------------------------------------------------------
 module Abortf is
 import Boolean
      type abortf is
          abort
        | noabort
      endtype

      function is_abort (AF:  abortf) : Bool is AF match abort endfunc
 endmod
========================================================================== *)

 type Control is NaturalNumber, Bit
      sorts  ctl
      opns
        ctl_iframe (*! constructor *) : Nat, Bit, Nat  ->  ctl
        ctl_sframe (*! constructor *) : Bit, Nat       ->  ctl
        ctl_uframe (*! constructor *) : Bit            ->  ctl
        undef         (*! constructor *) : ctl         ->  ctl
        is_undef_ctl  : ctl        ->  Bool
        nss           : ctl        ->  Nat
        nrr           : ctl        ->  Nat
        pff           : ctl        ->  Bit
      eqns
      forall  s, r : Nat , x : Bit , c : ctl
      ofsort Bool
        is_undef_ctl(undef(c)) = true of Bool;
        is_undef_ctl(ctl_iframe(s,x,r))  = false;
        is_undef_ctl(ctl_sframe(x,r))    = false;
        is_undef_ctl(ctl_uframe(x))      = false;
      ofsort Nat
        nss(ctl_iframe(s,x,r)) = s;
        nrr(ctl_iframe(s,x,r)) = r;
        nrr(ctl_sframe(x,r))   = r;
      ofsort Bit
        pff(ctl_iframe(s,x,r)) = x;
```

```
       pff(ctl_sframe(x,r))   = x;
       pff(ctl_uframe(x))     = x;
 endtype
(* ------------------------------------------------------------------------
 module Control is NaturalNumber, Bit
      type  ctl is
          ctl_iframe (S:Nat, X:Bit, R:Nat)
        | ctl_sframe (X:Bit, R:Nat)
        | ctl_uframe (X:Bit)
        | undef (C:ctl)
      endtype

     function is_undef_ctl (C: ctl) : Bool is C match undef(...) endfunc

      function nss (C: ctl) : Nat is select S in C endfunc

      function nrr (C: ctl) : Nat is select R in C endfunc

      function pff (C: ctl) : Bit is select X in C endfunc
 endmod
========================================================================== *)

 type Fcs is Boolean
      sorts fcs
      opns
        fc            (*! constructor *) :      -> fcs
        corrupt_fcs   (*! constructor *) : fcs  ->  fcs
        is_corrupt_fcs : fcs  ->  Bool
      eqns
      forall x: fcs
      ofsort Bool
        is_corrupt_fcs(fc)   = false;
        is_corrupt_fcs(corrupt_fcs(x)) = true of Bool;
 endtype
(* ------------------------------------------------------------------------
 module Fcs is Boolean
      type fcs is
          fc
        | corrupt_fcs(fcs)
      endtype

      function is_corrupt_fcs (F: fcs) : Bool is F match corrupt_fcs(...) endfunc
 endmod
========================================================================== *)

 type Frmrreason is Control,NaturalNumber,Bit
      sorts frmrreason
      opns
        nors        (*! constructor *) : -> frmrreason
        make_reason (*! constructor *) :
        ctl,Bit,Nat,Bit,Nat,Bit,Bit,Bit,Bit,Bit -> frmrreason
 endtype
(* ------------------------------------------------------------------------
 module Frmrreason is Control, NaturalNumber, Bit
```

```
     type frmrreason is
          nors
        | make_reason (ctl,Bit,Nat,Bit,Nat,Bit,Bit,Bit,Bit,Bit)
     endtype
 endmod
=================================================================== *)

 type Frametype is NaturalNumber, Packet, Adress, Flag, Control, Fcs,
       Frmrreason, Boolean, Infor, Corrlength, Lesslength
     sorts  frame
     opns
       make_iframe (*! constructor *)
               : flag, adress, ctl, packet, fcs, flag  -> frame
       make_rr     (*! constructor *)
               : flag, adress, ctl,fcs, flag  -> frame
       make_rnr    (*! constructor *)
               : flag, adress, ctl, fcs, flag -> frame
       make_rej    (*! constructor *)
               : flag, adress, ctl, fcs, flag -> frame
       make_sabm   (*! constructor *)
               : flag, adress, ctl, fcs, flag -> frame
       make_sabme  (*! constructor *)
               : flag, adress, ctl, fcs, flag -> frame
       make_disc   (*! constructor *)
               : flag, adress, ctl, fcs, flag -> frame
       make_ua     (*! constructor *)
               : flag, adress, ctl, fcs, flag -> frame
       make_dm     (*! constructor *)
               : flag, adress, ctl, fcs, flag -> frame
       make_frmr   (*! constructor *)
               : flag, adress, ctl, frmrreason, fcs, flag  -> frame
       get_ctl     : frame  ->  ctl
       get_flag1   : frame  ->  flag
       get_flag2   : frame  ->  flag
       get_adr     : frame  ->  adress
       get_packet  : frame  ->  packet
       get_info    : frame  ->  frmrreason
       get_fcs     : frame  ->  fcs
       is_i        : frame  ->  Bool
       is_rr       : frame  ->  Bool
       is_rnr      : frame  ->  Bool
       is_rej      : frame  ->  Bool
       is_sabm     : frame  ->  Bool
       is_sabme    : frame  ->  Bool
       is_disc     : frame  ->  Bool
       is_ua       : frame  ->  Bool
       is_dm       : frame  ->  Bool
       is_frmr     : frame  ->  Bool
       h           : frame  ->  Nat
       is_s        : frame  ->  Bool
       is_u        : frame  ->  Bool
     eqns
     forall f : frame, f1, f2 : flag, a : adress, c : ctl, inf : frmrreason,
            fcc : fcs, p : packet
```

```
      ofsort Nat
        h(make_iframe(f1,a,c,p,fcc,f2)) = 0;
        h(make_rr(f1,a,c,fcc,f2))       = Succ(0) ;
        h(make_rnr(f1,a,c,fcc,f2))      = Succ(Succ(0));
        h(make_rej(f1,a,c,fcc,f2))      = Succ(Succ(Succ(0)));
        h(make_sabm(f1,a,c,fcc,f2))     = Succ(Succ(Succ(Succ(0))));
        h(make_sabme(f1,a,c,fcc,f2))    = Succ(h(make_sabm(f1,a,c,fcc,f2))) ;
        h(make_disc(f1,a,c,fcc,f2))     = Succ(h(make_sabme(f1,a,c,fcc,f2)));
        h(make_ua(f1,a,c,fcc,f2))       = Succ(h(make_disc(f1,a,c,fcc,f2)));
        h(make_dm(f1,a,c,fcc,f2))       = Succ(h(make_ua(f1,a,c,fcc,f2)));
        h(make_frmr(f1,a,c,inf,fcc,f2))     = Succ(h(make_dm(f1,a,c,fcc,f2)));

      ofsort Bool
        is_i(f) = (h(f) eq 0);
        is_rr(f) = (h(f) eq Succ(0));
        is_rnr(f) = (h(f) eq Succ(Succ(0)));
        is_rej(f) = (h(f) eq Succ(Succ(Succ(0))));
        is_sabm(f) = (h(f) eq Succ(Succ(Succ(Succ(0)))));
        is_sabme(f) = (h(f) eq Succ(Succ(Succ(Succ(Succ(0))))));
        is_disc(f) = (h(f) eq Succ(Succ(Succ(Succ(Succ(Succ(0)))))));
        is_ua(f) = (h(f) eq Succ(Succ(Succ(Succ(Succ(Succ(Succ(0))))))));
        is_dm(f)= (h(f) eq (Succ(Succ(0)) ** Succ(Succ(Succ(0)))));
        is_frmr(f) = (h(f) eq (Succ(Succ(Succ(0))) * Succ(Succ(Succ(0)))));
        is_s(f) = (is_rr(f) or is_rnr(f)) or (is_rej(f)) ;
        is_u(f) = (((is_sabm(f)) or (is_sabme(f))) or (is_disc(f)))
or  (((is_ua(f)) or (is_dm(f))) or (is_frmr(f)))

      ofsort ctl
        get_ctl(make_iframe(f1,a,c,p,fcc,f2)) = c;
        get_ctl(make_rr(f1,a,c,fcc,f2)) = c;
        get_ctl(make_rnr(f1,a,c,fcc,f2)) = c;
        get_ctl(make_rej(f1,a,c,fcc,f2)) = c;
        get_ctl(make_sabm(f1,a,c,fcc,f2)) = c;
        get_ctl(make_sabme(f1,a,c,fcc,f2)) = c;
        get_ctl(make_disc(f1,a,c,fcc,f2)) = c;
        get_ctl(make_ua(f1,a,c,fcc,f2)) = c;
        get_ctl(make_dm(f1,a,c,fcc,f2)) = c;
        get_ctl(make_frmr(f1,a,c,inf,fcc,f2)) = c;
      ofsort flag
        get_flag1(make_iframe(f1,a,c,p,fcc,f2)) = f1;
        get_flag1(make_rr(f1,a,c,fcc,f2)) = f1;
        get_flag1(make_rnr(f1,a,c,fcc,f2)) = f1;
        get_flag1(make_rej(f1,a,c,fcc,f2)) = f1;
        get_flag1(make_sabm(f1,a,c,fcc,f2)) = f1;
        get_flag1(make_sabme(f1,a,c,fcc,f2)) = f1;
        get_flag1(make_disc(f1,a,c,fcc,f2)) = f1;
        get_flag1(make_ua(f1,a,c,fcc,f2)) = f1;
        get_flag1(make_dm(f1,a,c,fcc,f2)) = f1;
        get_flag1(make_frmr(f1,a,c,inf,fcc,f2)) = f1;
        get_flag2(make_iframe(f1,a,c,p,fcc,f2)) = f2;
        get_flag2(make_rr(f1,a,c,fcc,f2)) = f2;
        get_flag2(make_rnr(f1,a,c,fcc,f2)) = f2;
        get_flag2(make_rej(f1,a,c,fcc,f2)) = f2;
        get_flag2(make_sabm(f1,a,c,fcc,f2)) = f2;
        get_flag2(make_sabme(f1,a,c,fcc,f2)) = f2;
        get_flag2(make_disc(f1,a,c,fcc,f2)) = f2;
        get_flag2(make_ua(f1,a,c,fcc,f2)) = f2;
        get_flag2(make_dm(f1,a,c,fcc,f2)) = f2;
        get_flag2(make_frmr(f1,a,c,inf,fcc,f2)) = f2;
      ofsort adress
        get_adr(make_iframe(f1,a,c,p,fcc,f2)) = a;
        get_adr(make_rr(f1,a,c,fcc,f2)) = a;
        get_adr(make_rnr(f1,a,c,fcc,f2)) = a;
        get_adr(make_rej(f1,a,c,fcc,f2)) = a;
        get_adr(make_sabm(f1,a,c,fcc,f2)) = a;
        get_adr(make_sabme(f1,a,c,fcc,f2)) = a;
        get_adr(make_disc(f1,a,c,fcc,f2)) = a;
        get_adr(make_ua(f1,a,c,fcc,f2)) = a;
        get_adr(make_dm(f1,a,c,fcc,f2)) = a;
        get_adr(make_frmr(f1,a,c,inf,fcc,f2)) = a;
      ofsort ctl
        get_ctl(make_iframe(f1,a,c,p,fcc,f2)) = c;
        get_ctl(make_rr(f1,a,c,fcc,f2)) = c;
        get_ctl(make_rnr(f1,a,c,fcc,f2)) = c;
        get_ctl(make_rej(f1,a,c,fcc,f2)) = c;
        get_ctl(make_sabm(f1,a,c,fcc,f2)) = c;
        get_ctl(make_sabme(f1,a,c,fcc,f2)) = c;
        get_ctl(make_disc(f1,a,c,fcc,f2)) = c;
        get_ctl(make_ua(f1,a,c,fcc,f2)) = c;
        get_ctl(make_dm(f1,a,c,fcc,f2)) = c;
        get_ctl(make_frmr(f1,a,c,inf,fcc,f2)) = c;
      ofsort packet
        get_packet(make_iframe(f1,a,c,p,fcc,f2)) = p;
      ofsort fcs
        get_fcs(make_iframe(f1,a,c,p,fcc,f2)) = fcc;
        get_fcs(make_rr(f1,a,c,fcc,f2)) = fcc;
        get_fcs(make_rnr(f1,a,c,fcc,f2)) = fcc;
        get_fcs(make_rej(f1,a,c,fcc,f2)) = fcc;
        get_fcs(make_sabm(f1,a,c,fcc,f2)) = fcc;
        get_fcs(make_sabme(f1,a,c,fcc,f2)) = fcc;
        get_fcs(make_disc(f1,a,c,fcc,f2)) = fcc;
        get_fcs(make_ua(f1,a,c,fcc,f2)) = fcc;
        get_fcs(make_dm(f1,a,c,fcc,f2)) = fcc;
        get_fcs(make_frmr(f1,a,c,inf,fcc,f2)) = fcc;
      ofsort frmrreason
        get_info(make_frmr(f1,a,c,inf,fcc,f2)) = inf;
 endtype
(* ----------------------------------------------------------------------
 module Frametype is NaturalNumber, Packet, Adress, Flag, Control, Fcs,
       Frmrreason, Boolean, Infor, Corrlength, Lesslength

      type  frame is
          make_iframe (F1: flag,A: adress,C: ctl,P: packet,FCC: fcs,F2: flag)
        | make_rr     (F1: flag,A: adress,C: ctl,FCC: fcs,F2: flag)
        | make_rnr    (F1: flag,A: adress,C: ctl,FCC: fcs,F2: flag)
        | make_rej    (F1: flag,A: adress,C: ctl,FCC: fcs,F2: flag)
        | make_sabm   (F1: flag,A: adress,C: ctl,FCC: fcs,F2: flag)
        | make_sabme  (F1: flag,A: adress,C: ctl,FCC: fcs,F2: flag)
        | make_disc   (F1: flag,A: adress,C: ctl,FCC: fcs,F2: flag)
        | make_ua     (F1: flag,A: adress,C: ctl,FCC: fcs,F2: flag)
        | make_dm     (F1: flag,A: adress,C: ctl,FCC: fcs,F2: flag)
        | make_frmr   (F1: flag,A: adress,C: ctl,INF: frmrreason,FCC: fcs,F2: flag)
      endtype

     function get_ctl    (F: frame) : ctl is select C in  F endfunc

     function get_flag1  (F: frame) : flag is select F1 in F endfunc

     function get_flag2 (F: frame) : flag is select F2 in F endfunc

     function get_adr (F: frame) : adress is select A in F endfunc

     function get_packet (F: frame) : packet is select P in F endfunc

     function get_info (F: frame) : frmrreason is select INF in F endfunc

     function get_fcs (F: frame) : fcs is select FCC in F endfunc

/* no translation is needed for function h */

     function is_i (f: frame) : Bool is f match make_iframe (...) endfunc

     function is_rr (f: frame) : Bool is f match make_rr (...) endfunc

     function is_rnr (f: frame) : Bool is f match make_rnr (...) endfunc

     function is_rej (f: frame) : Bool is f match make_rej (...) endfunc

     function is_sabm (f: frame) : Bool is f match make_sabm (...) endfunc

     function is_sabme (f: frame) : Bool is f match make_sabme (...) endfunc

     function is_disc (f: frame) : Bool is f match make_disc (...) endfunc

     function is_ua (f: frame) : Bool is  f match make_ua (...) endfunc

     function is_dm (f: frame) : Bool is  f match make_dm (...) endfunc

     function is_frmr (f: frame) : Bool is f match make_frmr (...) endfunc

     function is_s  (f: frame) : Bool is (is_rr(f) or is_rnr(f)) or (is_rej(f)) endfunc
/*
     function is_s  (f: frame) : Bool is
     (f match make_rnr (...)) or (f match make_rr (...)) or (f match make_rej (...))
     endfunc
     function is_u  (f: frame) : Bool is
        (((is_sabm(f)) or (is_sabme(f))) or (is_disc(f)))
        or  (((is_ua(f)) or (is_dm(f))) or (is_frmr(f)))
     endfunc
*/
     function is_u  (f: frame) : Bool is
```

```
       (f match make_sabm (...)) or (f match make_sabme (...)) or (f match make_disc (...))
       or (f match make_ua (...)) or (f match make_dm (...)) or (f match make_frmr (...))
    endfunc
 endmod
========================================================================= *)

 type Extframetype is NaturalNumber,Frametype,Infor, Corrlength,
       Lesslength, Abortf, Boolean,Mode
     sorts  eframe
     opns
        _mod_ : Nat,Nat         -> Nat
       _-_   : Nat,Nat         -> Nat
       m (*! constructor *) : frame,inforr,corrlength,lesslength,abortf -> eframe
       collis    : eframe,eframe -> Bool
       resp      : eframe -> Bool
       is_valid  : eframe -> Bool
       is_reject : eframe -> Bool
       not_abort : eframe -> Bool
       correct   : eframe -> Bool
       incorrect : eframe -> Bool
       flg1      : eframe -> flag
       flg2      : eframe -> flag
       pack      : eframe -> packet
       adrs      : eframe -> adress
       ctrl      : eframe -> ctl
       fcss      : eframe -> fcs
       reasfrmr  : eframe -> frmrreason
       ns        : eframe -> Nat
       nr        : eframe -> Nat
       pf        : eframe -> Bit
       get_infor : eframe -> inforr
       get_c     : eframe -> corrlength
       get_l     : eframe -> lesslength
       get_f     : eframe -> frame
       get_a     : eframe -> abortf
       is_inforr : eframe -> Bool
       corrl     : eframe -> Bool
       _eq_      : eframe,eframe -> Bool
               _==_      : eframe,eframe -> Bool
       notvalid_nrf      : eframe,Nat,Nat,Nat -> Bool
               makecorruptadr    : eframe -> eframe
               makeundefctl      : eframe -> eframe
               makecorruptfc     : eframe -> eframe
               makeinfor         : eframe -> eframe
               makenotcorrlength : eframe -> eframe
               makelesslength    : eframe -> eframe

     eqns
       forall f: eframe ,f1 : eframe, fr : frame, a : abortf,
          ir : inforr , c : corrlength, l : lesslength,
       vs ,lu,x,y, k : Nat
       ofsort flag
         flg1(m(fr,ir,c,l,a)) = get_flag1(fr);
         flg2(m(fr,ir,c,l,a)) = get_flag2(fr);
```

```
ofsort packet
  pack(m(fr,ir,c,l,a)) = get_packet(fr);
ofsort adress
  adrs(m(fr,ir,c,l,a)) = get_adr(fr);
ofsort ctl
  ctrl(m(fr,ir,c,l,a)) = get_ctl(fr);
ofsort fcs
  fcss(m(fr,ir,c,l,a)) = get_fcs(fr);
ofsort frmrreason
  reasfrmr(m(fr,ir,c,l,a)) = get_info(fr);
ofsort Nat
  ns(m(fr,ir,c,l,a)) = nss(get_ctl(fr));
  nr(m(fr,ir,c,l,a)) = nrr(get_ctl(fr));
ofsort Bit
  pf(m(fr,ir,c,l,a)) = pff(get_ctl(fr));
ofsort abortf
  get_a(m(fr,ir,c,l,a)) = a;
ofsort inforr
  get_infor(m(fr,ir,c,l,a)) = ir;
ofsort corrlength
  get_c(m(fr,ir,c,l,a)) = c;
ofsort lesslength
  get_l(m(fr,ir,c,l,a)) = l;
ofsort frame
  get_f(m(fr,ir,c,l,a)) = fr;
ofsort Bool
  (h(get_f(f)) eq h(get_f(f1))) => collis(f,f1)= true of Bool;
  (h(get_f(f)) ne h(get_f(f1))) => collis(f,f1)= false;
  (is_sabm(get_f(f)) or is_sabme(get_f(f))) => resp(f) = false;
  is_disc(get_f(f)) => resp(f) = true of Bool;
  corrl(f) = is_correct(get_c(f));
  is_inforr(f) = is_infor(get_infor(f));
  is_i(get_f(f))  => is_reject(f) = (is_undef_ctl(ctrl(f)) )
      or (not(is_correct(get_c(f))));
  not(is_i(get_f(f))) => is_reject(f) = ((is_undef_ctl(ctrl(f))) or
  (not(is_correct(get_c(f)))) ) or is_infor(get_infor(f));
  is_valid(f) = not(((is_corrupt_adr(adrs(f))) or
       (is_corrupt_fcs(fcss(f)))) or
  (((not(is_flag(get_flag1(get_f(f))))) or (not(is_flag
       (get_flag2(get_f(f)))))) or (not(is_not_less(get_l(f))))));
  not_abort(f) = not(is_abort(get_a(f)));
  correct(f) = (is_valid(f) and (not(is_reject(f)))) and not_abort(f);

  incorrect(f) = (not(is_valid(f))) or (is_reject(f) or
       (not(not_abort(f))));
  f eq f1 = h(get_f(f)) eq h(get_f(f1)) ;
          f = f1 => f == f1 = true of Bool;
          f == f1 = false;
  notvalid_nrf(f,vs,lu,k) = not((((nr(f) ge lu) or (nr(f) le
       (vs mod k))) and ((vs mod k) lt lu)) or (((nr(f) ge lu)
   and (nr(f) le (vs mod k))) and ((vs mod k) ge lu)));
ofsort Nat
  x - x            = 0;
  Succ(x) - y      = Succ(x - y);
```

```
x mod 0           = x;
x lt y => x mod y = x;
x mod y           = (x - y) mod y;

      ofsort eframe
        is_i(get_f(f)) => makecorruptadr(f) = m(make_iframe(flg1(f),
            corrupt_adr(adrs(f)), ctrl(f), pack(f), fcss(f),
            flg2(f)), get_infor(f), get_c(f), get_l(f), get_a(f));
        is_rr(get_f(f)) => makecorruptadr(f) = m(make_rr(flg1(f),
            corrupt_adr(adrs(f)), ctrl(f), fcss(f),
            flg2(f)), get_infor(f), get_c(f), get_l(f), get_a(f));
        is_rnr(get_f(f)) => makecorruptadr(f) = m(make_rnr(flg1(f),
            corrupt_adr(adrs(f)), ctrl(f), fcss(f),
            flg2(f)), get_infor(f), get_c(f), get_l(f), get_a(f));
        is_rej(get_f(f)) => makecorruptadr(f) = m(make_rej(flg1(f),
            corrupt_adr(adrs(f)), ctrl(f), fcss(f),
            flg2(f)), get_infor(f), get_c(f), get_l(f), get_a(f));
        is_sabm(get_f(f)) => makecorruptadr(f) = m(make_sabm(flg1(f),
            corrupt_adr(adrs(f)), ctrl(f), fcss(f),
            flg2(f)), get_infor(f), get_c(f), get_l(f), get_a(f));
        is_sabme(get_f(f)) =>
            makecorruptadr(f) = m(make_sabme(flg1(f),
            corrupt_adr(adrs(f)), ctrl(f), fcss(f),
            flg2(f)), get_infor(f), get_c(f), get_l(f), get_a(f));
        is_disc(get_f(f)) => makecorruptadr(f) = m(make_disc(flg1(f),
            corrupt_adr(adrs(f)), ctrl(f), fcss(f),
            flg2(f)), get_infor(f), get_c(f), get_l(f), get_a(f));
        is_ua(get_f(f)) => makecorruptadr(f) = m(make_ua(flg1(f),
            corrupt_adr(adrs(f)), ctrl(f), fcss(f),
            flg2(f)), get_infor(f), get_c(f), get_l(f), get_a(f));
        is_dm(get_f(f)) => makecorruptadr(f) = m(make_dm(flg1(f),
            corrupt_adr(adrs(f)), ctrl(f), fcss(f),
            flg2(f)), get_infor(f), get_c(f), get_l(f), get_a(f));
        is_frmr(get_f(f)) => makecorruptadr(f) = m(make_frmr(flg1(f),
            corrupt_adr(adrs(f)), ctrl(f), reasfrmr(f), fcss(f),
            flg2(f)), get_infor(f), get_c(f), get_l(f), get_a(f));

        is_i(get_f(f)) => makeundefctl(f) = m(make_iframe(flg1(f),
            adrs(f), undef(ctrl(f)), pack(f), fcss(f),
            flg2(f)), get_infor(f), get_c(f), get_l(f), get_a(f));
        is_rr(get_f(f)) => makeundefctl(f) = m(make_rr(flg1(f),
            adrs(f), undef(ctrl(f)), fcss(f),
            flg2(f)), get_infor(f), get_c(f), get_l(f), get_a(f));
        is_rnr(get_f(f)) => makeundefctl(f) = m(make_rnr(flg1(f),
            adrs(f), undef(ctrl(f)), fcss(f),
            flg2(f)), get_infor(f), get_c(f), get_l(f), get_a(f));
        is_rej(get_f(f)) => makeundefctl(f) = m(make_rej(flg1(f),
            adrs(f), undef(ctrl(f)), fcss(f),
            flg2(f)), get_infor(f), get_c(f), get_l(f), get_a(f));
        is_sabm(get_f(f)) => makeundefctl(f) = m(make_sabm(flg1(f),
            adrs(f), undef(ctrl(f)), fcss(f),
            flg2(f)), get_infor(f), get_c(f), get_l(f), get_a(f));
        is_sabme(get_f(f)) => makeundefctl(f) = m(make_sabme(flg1(f),
            adrs(f), undef(ctrl(f)), fcss(f), flg2(f)), get_infor(f),
            get_c(f), get_l(f), get_a(f));
        is_disc(get_f(f)) => makeundefctl(f) = m(make_disc(flg1(f),
            adrs(f), undef(ctrl(f)), fcss(f),
            flg2(f)), get_infor(f), get_c(f), get_l(f), get_a(f));
        is_ua(get_f(f)) => makeundefctl(f) = m(make_ua(flg1(f),
            adrs(f), undef(ctrl(f)), fcss(f), flg2(f)), get_infor(f),
            get_c(f), get_l(f), get_a(f));
        is_dm(get_f(f)) => makeundefctl(f) = m(make_dm(flg1(f),
            adrs(f), undef(ctrl(f)), fcss(f),
            flg2(f)), get_infor(f), get_c(f), get_l(f), get_a(f));
        is_frmr(get_f(f)) => makeundefctl(f) = m(make_frmr(flg1(f),
            adrs(f), undef(ctrl(f)), reasfrmr(f), fcss(f),
            flg2(f)), get_infor(f), get_c(f), get_l(f), get_a(f));

        is_i(get_f(f)) => makecorruptfc(f) = m(make_iframe(flg1(f),
            adrs(f), ctrl(f), pack(f), corrupt_fcs(fcss(f)),
            flg2(f)), get_infor(f), get_c(f), get_l(f), get_a(f));
        is_rr(get_f(f)) => makecorruptfc(f) = m(make_rr(flg1(f),
            adrs(f), ctrl(f), corrupt_fcs(fcss(f)),
            flg2(f)), get_infor(f), get_c(f), get_l(f), get_a(f));
        is_rnr(get_f(f)) => makecorruptfc(f) = m(make_rnr(flg1(f),
            adrs(f), ctrl(f), corrupt_fcs(fcss(f)),
            flg2(f)), get_infor(f), get_c(f), get_l(f), get_a(f));
        is_rej(get_f(f)) => makecorruptfc(f) = m(make_rej(flg1(f),
            adrs(f), ctrl(f), corrupt_fcs(fcss(f)),
            flg2(f)), get_infor(f), get_c(f), get_l(f), get_a(f));
        is_sabm(get_f(f)) => makecorruptfc(f) = m(make_sabm(flg1(f),
            adrs(f), ctrl(f), corrupt_fcs(fcss(f)),
            flg2(f)), get_infor(f), get_c(f), get_l(f), get_a(f));
        is_sabme(get_f(f)) =>
            makecorruptfc(f) = m(make_sabme(flg1(f),
            adrs(f), ctrl(f), corrupt_fcs(fcss(f)),
            flg2(f)), get_infor(f), get_c(f), get_l(f), get_a(f));
        is_disc(get_f(f)) => makecorruptfc(f) = m(make_disc(flg1(f),
            adrs(f), ctrl(f), corrupt_fcs(fcss(f)),
            flg2(f)), get_infor(f), get_c(f), get_l(f), get_a(f));
        is_ua(get_f(f)) => makecorruptfc(f) = m(make_ua(flg1(f),
            adrs(f), ctrl(f), corrupt_fcs(fcss(f)),
            flg2(f)), get_infor(f), get_c(f), get_l(f), get_a(f));
        is_dm(get_f(f)) => makecorruptfc(f) = m(make_dm(flg1(f),
            adrs(f), ctrl(f), corrupt_fcs(fcss(f)),
            flg2(f)), get_infor(f), get_c(f), get_l(f), get_a(f));
        is_frmr(get_f(f)) => makecorruptfc(f) = m(make_frmr(flg1(f),
            adrs(f), ctrl(f), reasfrmr(f), corrupt_fcs(fcss(f)),
            flg2(f)), get_infor(f), get_c(f), get_l(f), get_a(f));
is_i(get_f(f)) => makeinfor(f) = f;
        not(is_i(get_f(f))) => makeinfor(f) = m(get_f(f), infor,
    get_c(f), get_l(f), get_a(f));
        makenotcorrlength(f) = m(get_f(f), get_infor(f), incorrect,
    get_l(f), get_a(f));
makelesslength(f) = m(get_f(f), get_infor(f), get_c(f), less,
    get_a(f));
```

```
 endtype
(* ------------------------------------------------------------------------
 module Extframetype is NaturalNumber,Frametype,Infor, Corrlength,
        Lesslength, Abortf, Boolean,Mode
      type  eframe is
          m (FR:frame,IR: inforr,C: corrlength,L: lesslength,A: abortf)
      endtype

    function get_f (F: eframe) : frame is select FR in F endfunc

    function get_c (F: eframe) : corrlength is select C in F endfunc

    function get_l (F: eframe) : lesslength  is select L in F endfunc

    function get_a (F: eframe) : abortf is select A in F endfunc

    function collis (EF1: eframe,EF2: eframe) : Bool is
        get_f(EF1) eq get_f(EF2)
    endfunc

    function resp (EF: eframe) : Bool is
        case (select FR in EF) in
            make_sabm(...)
        | make_sabme -> false
        | make_disc(...) ->  true of Bool
        endcase
    endfunc
/*
    function resp (EF: eframe) : Bool is
        case true in
            true when (is_sabm(get_f(EF)) or is_sabme(get_f(EF)))       -> false
          | true when is_disc(get_f(EF)) ->  true of Bool
        endcase
    endfunc
*/
    function is_valid  (F: eframe) : Bool is
       not((is_corrupt_adr(adrs(F)) or
            is_corrupt_fcs(fcss(f)))
         or ((not(is_flag(get_flag1(get_f(f)))) or
              not(is_flag(get_flag2(get_f(f)))))
         or not(is_not_less(get_l(f)))))
    endfunc

    function is_reject (F: eframe) : Bool is
        if is_i(get_f(F)) then
                is_undef_ctl(ctrl(F)) or not(is_correct(get_c(f)))
        else
                (is_undef_ctl(ctrl(f)) or not(is_correct(get_c(f))))  or is_infor(get_infor(f))
        endif
    endfunc

    function not_abort (F: eframe) : Bool is not(is_abort(get_a(F))) endfunc

    function correct (F: eframe) : Bool is
```

```
        (is_valid(f) and (not(is_reject(f)))) and not_abort(f)
    endfunc

    function incorrect (F: eframe) : Bool is
          (not(is_valid(f))) or (is_reject(f) or (not(not_abort(f))))
    endfunc

    function flg1 (F: eframe) : flag is get_flag1(select FR in F) endfunc

    function flg2 (F: eframe) : flag is get_flag2(select FR in F) endfunc

    function pack (F: eframe) : packet is get_packet(select FR in F) endfunc

    function adrs (F: eframe) : adress is get_adr(select FR in F) endfunc

    function ctrl (F: eframe) : ctl is get_ctl(select FR in F) endfunc

    function fcss (F: eframe) : fcs is get_fcs(select FR in F) endfunc

    function reasfrmr (F: eframe) : frmrreason is get_info(select FR in F) endfunc

    function ns (F: eframe) : Nat is nns(get_ctl(select FR in F)) endfunc

    function nr (F: eframe) : Nat is nrr(get_ctl(select FR in F)) endfunc

    function pf (F: eframe) : Bit is pff(get_ctl(select FR in F)) endfunc

    function get_infor (F: eframe) : inforr is select IR in F endfunc

    function is_inforr (F: eframe) : Bool is is_infor(select IR in F) endfunc

    function corrl (F: eframe) : Bool is is_correct(select C in F) endfunc

    function notvalid_nrf (F: eframe,VS: Nat,LU: Nat,K: Nat) : Bool is
        not((((nr(F) ge LU) or (nr(F) le (VS mod K))) and ((VS mod K) lt LU)) or (((nr(F) ge LU)
           and (nr(F) le (VS mod K))) and ((VS mod K) ge LU)))
    endfunc

    function _eq1_ (f1: eframe, f2: eframe) : Bool is
        get_f(f1) eq get_f(f2)
    endfunc

    function makecorruptadr (F: eframe) : eframe is
        case (select FR in F) in
          make_iframe (...) -> m(make_iframe(flg1(F),
                corrupt_adr(adrs(F)), ctrl(F), pack(F), fcss(F),
                flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
        | make_rr (...) -> m(make_rr(flg1(F),
                corrupt_adr(adrs(F)), ctrl(F), fcss(F),
                flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
        | make_rnr (...) -> m(make_rnr(flg1(F),
                corrupt_adr(adrs(F)), ctrl(F), fcss(F),
                flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
        | make_rej (...) -> m(make_rej(flg1(F),
            corrupt_adr(adrs(F)), ctrl(F), fcss(F),
            flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
      | make_sabm (...) -> m(make_sabm(flg1(F),
            corrupt_adr(adrs(F)), ctrl(F), fcss(F),
            flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
      | make_sabme (...) -> m(make_sabme(flg1(F),
            corrupt_adr(adrs(F)), ctrl(F), fcss(F),
            flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
      | make_disc (...) -> m(make_disc(flg1(F),
            corrupt_adr(adrs(F)), ctrl(F), fcss(F),
            flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
      | make_ua (...) -> m(make_ua(flg1(F),
            corrupt_adr(adrs(F)), ctrl(F), fcss(F),
            flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
      | make_dm (...) -> m(make_dm(flg1(F),
            corrupt_adr(adrs(F)), ctrl(F), fcss(F),
            flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
      | make_frmr(...) -> m(make_frmr(flg1(F),
            corrupt_adr(adrs(F)), ctrl(F), reasfrmr(F), fcss(F),
            flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
     endcase
 endfunc

  function makeundefctl (F: eframe) : eframe is
     case (select FR in F) in
       make_iframe (...) -> m(make_iframe(flg1(F),
            adrs(F), undef(ctrl(F)), pack(F), fcss(F),
            flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
      | make_rr (...) -> m(make_rr(flg1(F),
            adrs(F), undef(ctrl(F)), pack(F), fcss(F),
            flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
      | make_rnr (...) -> m(make_rnr(flg1(F),
            adrs(F), undef(ctrl(F)), pack(F), fcss(F),
            flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
      | make_rej (...) -> m(make_rej(flg1(F),
            adrs(F), undef(ctrl(F)), pack(F), fcss(F),
            flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
      | make_sabm (...) -> m(make_sabm(flg1(F),
            adrs(F), undef(ctrl(F)), pack(F), fcss(F),
            flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
      | make_sabme (...) -> m(make_sabme(flg1(F),
            adrs(F), undef(ctrl(F)), pack(F), fcss(F),
            flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
      | make_disc (...) -> m(make_disc(flg1(F),
            adrs(F), undef(ctrl(F)), pack(F), fcss(F),
            flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
      | make_ua (...) -> m(make_ua(flg1(F),
            adrs(F), undef(ctrl(F)), pack(F), fcss(F),
            flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
      | make_dm (...) -> m(make_dm(flg1(F),
            adrs(F), undef(ctrl(F)), pack(F), fcss(F),
            flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
      | make_frmr(...) -> m(make_frmr(flg1(F),
            adrs(F), undef(ctrl(F)), pack(F), fcss(F),
                flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
        endcase
    endfunc

    function makecorruptfc     (F: eframe) : eframe is
        case (select FR in F) in
          make_iframe (...) -> m(make_iframe(flg1(F),
                adrs(F), ctrl(F), pack(F), corrupt_fcs(fcss(F)),
                flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
        | make_rr (...) -> m(make_rr(flg1(F),
                adrs(F), ctrl(F), pack(F), corrupt_fcs(fcss(F)),
                flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
        | make_rnr (...) -> m(make_rnr(flg1(F),
                adrs(F), ctrl(F), pack(F), corrupt_fcs(fcss(F)),
                flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
        | make_rej (...) -> m(make_rej(flg1(F),
                adrs(F), ctrl(F), pack(F), corrupt_fcs(fcss(F)),
                flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
        | make_sabm (...) -> m(make_sabm(flg1(F),
                adrs(F), ctrl(F), pack(F), corrupt_fcs(fcss(F)),
                flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
        | make_sabme (...) -> m(make_sabme(flg1(F),
                adrs(F), ctrl(F), pack(F), corrupt_fcs(fcss(F)),
                flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
        | make_disc (...) -> m(make_disc(flg1(F),
                adrs(F), ctrl(F), pack(F), corrupt_fcs(fcss(F)),
                flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
        | make_ua (...) -> m(make_ua(flg1(F),
                adrs(F), ctrl(F), pack(F), corrupt_fcs(fcss(F)),
                flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
        | make_dm (...) -> m(make_dm(flg1(F),
                adrs(F), ctrl(F), pack(F), corrupt_fcs(fcss(F)),
                flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
        | make_frmr(...) -> m(make_frmr(flg1(F),
                adrs(F), ctrl(F), pack(F), corrupt_fcs(fcss(F)),
                flg2(F)), get_infor(F), get_c(F), get_l(F), get_a(F))
        endcase
    endfunc

    function makeinfor  (F: eframe) : eframe is
        if is_i(get_f(F)) then F
        else m(get_f(F), infor, get_c(F), get_l(F), get_a(F))
    endfunc

    function makenotcorrlength (F: eframe) : eframe is
        m(get_f(F), get_infor(F), incorrect,get_l(F), get_a(F))
    endfunc

    function makelesslength (F: eframe) : eframe is
        m(get_f(F), get_infor(F), get_c(F), less, get_a(F))
    endfunc
 endmod
========================================================================= *)
```

```
 type primitive is Packet,Boolean,NaturalNumber
      sorts primitive
      opns
        dl_est_req (*! constructor *) :      -> primitive
        dl_dis_req (*! constructor *) :      -> primitive
        dl_dis_cnf (*! constructor *) :      -> primitive
        dl_est_ind (*! constructor *) :      -> primitive
        dl_est_cnf (*! constructor *) :      -> primitive
        dl_dis_ind (*! constructor *) :      -> primitive
        dl_data_req (*! constructor *) : packet  -> primitive
        dl_data_ind (*! constructor *) : packet  -> primitive
        eval          : primitive -> Nat
        is_dl_est_req : primitive -> Bool
        is_dl_dis_req : primitive -> Bool
        is_dl_dis_cnf : primitive -> Bool
        is_dl_est_ind : primitive -> Bool
        is_dl_est_cnf : primitive -> Bool
        is_dl_dis_ind : primitive -> Bool
        is_dl_data_req : primitive -> Bool
        is_dl_data_ind : primitive -> Bool
        get_pack       : primitive -> packet
      eqns
      forall p : primitive,pa : packet
      ofsort packet
        get_pack(dl_data_req(pa)) = pa;
      ofsort Nat
        eval(dl_est_req) = 0;
        eval(dl_dis_req) = Succ(0);
        eval(dl_dis_cnf) = Succ(Succ(0));
        eval(dl_est_ind) = Succ(Succ(Succ(0)));
        eval(dl_est_cnf) = Succ(Succ(Succ(Succ(0))));
        eval(dl_dis_ind) = Succ(Succ(Succ(Succ(Succ(0)))));
        eval(dl_data_req(pa)) = Succ(Succ(Succ(Succ(Succ(Succ(0))))));
        eval(dl_data_ind(pa)) = Succ(Succ(Succ(Succ(Succ(Succ(Succ(0)))))));
      ofsort Bool
        is_dl_est_req(p) = (eval(p) eq 0);
        is_dl_dis_req(p) = (eval(p) eq Succ(0));
        is_dl_dis_cnf(p) = (eval(p) eq Succ(Succ(0)));
        is_dl_est_ind(p) = (eval(p) eq Succ(Succ(Succ(0))));
        is_dl_est_cnf(p) = (eval(p) eq Succ(Succ(Succ(Succ(0)))));
        is_dl_dis_ind(p) = (eval(p) eq Succ(Succ(Succ(Succ(Succ(0))))));
        is_dl_data_req(p) = (eval(p) eq Succ(Succ(Succ(Succ(Succ(Succ(0)))))))
;
        is_dl_data_ind(p) = (eval(p) eq Succ(Succ(Succ(Succ(Succ(Succ
                            (Succ(0))))))));
 endtype
(* -----------------------------------------------------------------------
 module primitive is  Packet, Boolean, NaturalNumber
      type primitive is
          dl_est_req
        | dl_dis_req
        | dl_dis_cnf
        | dl_est_ind
        | dl_est_cnf
        | dl_dis_ind
        | dl_data_req (PA:packet)
        | dl_data_ind (packet)
      endtype

/* no translation is needed for function eval */

    function is_dl_est_req (P: primitive) : Bool is P match dl_est_req endfunc

    function is_dl_dis_req (P: primitive) : Bool is P match dl_dis_req endfunc

    function is_dl_dis_cnf (P: primitive) : Bool is P match dl_dis_cnf endfunc

    function is_dl_est_ind (P: primitive) : Bool is P match dl_est_ind endfunc

    function is_dl_est_cnf (P: primitive) : Bool is P match dl_est_cnf endfunc

    function is_dl_dis_ind (P: primitive) : Bool is P match dl_dis_ind  endfunc

    function is_dl_data_req (P: primitive) : Bool is P match dl_data_req (...)  endfunc

    function is_dl_data_ind (P: primitive) : Bool is P match dl_data_ind (...)  endfunc

    function get_pack (P: primitive) : packet is select PA in P endfunc
 endmod
========================================================================== *)

 type Queue  is Boolean
      formalsorts elt
      sorts queue
      opns
        empty    (*! constructor *) :            -> queue
        _+--_    : elt,queue  -> queue
        _--+_    (*! constructor *) : queue,elt   -> queue
        isempty  : queue          -> Bool
        firstone : queue          -> elt
        removefirst : queue       -> queue
      eqns
      forall pq,pq1,pq2 : queue , f , f1, f2 : elt
      ofsort Bool
        isempty(empty)   = true of Bool;
        (* isempty(f +-- pq) = false; *)
        isempty(pq --+ f) = false;
      ofsort elt
        firstone(pq --+ f) = f;
      ofsort queue
        f +-- empty      = empty --+ f;
        f1 +-- (pq --+ f2) = (f1 +-- pq) --+ f2;
        removefirst(pq --+ f) = pq;
 endtype
(* ------------------------------------------------------------------------
 module Queue  is Boolean
 parameter descr{sorts elt}
      type queue is
```

```
            empty
          | _--+_  (PQ: queue,E: elt)
      endtype

      function  _+--_  (E: elt,Q: queue) : queue is
        case Q in
                empty -> empty --+ E
              | ((PQ:=PQ: queue) --+ (E:=E2: elt)) -> (E+--PQ) --+ E2
        endcase
      endfunc

      function isempty  (Q: queue) : Bool is Q match empty endfunc

      function firstone (Q: queue) : elt is select E in Q endfunc

      function removefirst (Q: queue) : queue is select PQ in Q endfunc
 endmod
========================================================================== *)

 type Pacqueue is
      Queue actualizedby Packet using
      sortnames packet for elt
                pacqueue for queue
 endtype
(* -------------------------------------------------------------------------
 module Pacqueue is Queue [Packet:: sorts packet for elt]
                          {sorts pacqueue for queue}
 endmod
========================================================================== *)

 type Framequeue is
      Queue actualizedby Extframetype using
      sortnames eframe for elt
                fqueue for queue
 endtype
(* -------------------------------------------------------------------------
 module Framequeue is Queue[Extframetype:: sorts eframe for elt]
                        {sorts fqueue for queue}
 endmod
========================================================================== *)

 type Packe is Packet,NaturalNumber
      sorts pac
      opns
        mp (*! constructor *) : packet,Nat  -> pac
        snn : pac        -> Nat
        packe : pac      -> packet
      eqns
      forall s : Nat,p : packet
      ofsort Nat
        snn(mp(p,s)) = s;
      ofsort packet
        packe(mp(p,s)) = p;
 endtype
```

```
(* ---------------------------------------------------------------------------
 module Packe is Packet,NaturalNumber
      type pac is
        mp (PK: packet,S: Nat)
      endtype

      function snn (P: pac) : Nat is select S in P endfunc

      function packe (P: pac) : packet is select PK in P endfunc
 endmod
========================================================================= *)

 type Pqueue is
      Queue actualizedby Packe using
      sortnames pqueue for queue
                pac for elt
 endtype
(* ---------------------------------------------------------------------------
 module Pqueue is Queue[Packe:: sorts pac for elt]
                        {sorts pqueue for queue}
 endmod
========================================================================= *)


type VarOp is
(*
    Definition des operateurs de traitement des variables globales;
    Read: consultation de la valeur d'une variable
    Write: assignation d'une nouvelle valeur a cette variable
*)
sorts   varOp
opns    Read (*! constructor *) : -> varOp
        Write (*! constructor *) : -> varOp
endtype (* VarOp *)
(* ---------------------------------------------------------------------------
module VarOp is
        type   varOp is
               Read
             | Write
        endtype
endmod
========================================================================= *)

type GateOp is
(*
    Definition des operateurs sur les portes permettant de distinguer les
    entrees des sorties
*)
sorts   gateOp
opns    Input (*! constructor *) : -> gateOp
        Out (*! constructor *) : -> gateOp
endtype (* GateOp *)
(* ---------------------------------------------------------------------------
module GateOp is
```

```
        type   gateOp is
                Input
              | Out
        endtype
endmod
========================================================================== *)

type    DecimalConstants is NaturalNumber, DecNatRepr
opns    1, 6, 7, 8, 10, 128, 1080 : -> Nat
        TIMEOUT : -> Nat
        Pred : Nat -> Nat

eqns    forall x : Nat

        ofsort Nat

        Pred(Succ(x)) = x;
        Pred(0) = 0;
        TIMEOUT = 0;
        1 = Succ(0);
        6 = NatNum(Dec(6));
        7 = NatNum(Dec(7));
        8 = NatNum(Dec(8));
        10 = NatNum(Dec(1) ++ Dec(0));
        128 = NatNum(Dec(1) ++ Dec(2) ++ Dec(8));
        1080 = NatNum(Dec(1) ++ Dec(0) ++ Dec(8) ++ Dec(0));

endtype (* DecimalConstants *)
(* -----------------------------------------------------------------------------
/* no translation is needed for module DecimalConstants
        because numbers will be buil-in */
========================================================================== *)

type    wxyz is BitString, Bit
opns    w, x, y, z : BitString -> Bit

eqns    forall w, x, y, z : Bit

        ofsort Bit

(*       w(Bit(w) ++ (Bit(x) ++ (Bit(y) ++ Bit(z)))) = w;
        x(Bit(w) ++ (Bit(x) ++ (Bit(y) ++ Bit(z)))) = x;
        y(Bit(w) ++ (Bit(x) ++ (Bit(y) ++ Bit(z)))) = y;
        z(Bit(w) ++ (Bit(x) ++ (Bit(y) ++ Bit(z)))) = z;
        correct equations:
*)
        w(w + (x + (y + Bit(z)))) = w;
        x(w + (x + (y + Bit(z)))) = x;
        y(w + (x + (y + Bit(z)))) = y;
        z(w + (x + (y + Bit(z)))) = z;

endtype (* wxyz *)
(* -----------------------------------------------------------------------------
module    wxyz is BitString, Bit
```

```
function w (BS: BitString) : Bit is
  let (W:Bit) + ((X:Bit) + ((Y:Bit) + (Z:Bit)) = BS in W
endfunc

function x (BS: BitString) : Bit is
  let (W:Bit) + ((X:Bit) + ((Y:Bit) + (Z:Bit)) = BS in X
endfunc

function y (BS: BitString) : Bit is
  let (W:Bit) + ((X:Bit) + ((Y:Bit) + (Z:Bit)) = BS in Y
endfunc

function z (BS: BitString) : Bit is
  let (W:Bit) + ((X:Bit) + ((Y:Bit) + (Z:Bit)) = BS in Z
endfunc

endmod
============================================================================ *)

(****************************************************************************

        C O N S T R A I N T   D E F I N I T I O N S

 ***************************************************************************)

type Constraint is NaturalNumber, Boolean, Extframetype, Frametype, Packet,
    Infor, Corrlength, Lesslength, Abortf, Mode
     (* !! sorts constr *)
    opns
(* Trames "unnumbered *)
        DISC_S10, DISC_S11, DISC_S30, DISC_S31, DISC_S40,
        SABM_S10, SABM_S11, SABM_S30, SABM_S31, SABM_S40, SABM_S41,
        UA_S10, UA_S11, UA_S30, UA_S31,
        DM_S10, DM_S11, DM_S30, DM_S31 : -> eframe
        UA_S1F, DM_S1F : Bit -> eframe
        FRMR_S10, FRMR_S11, FRMR_S30, FRMR_S31,
        FRMR_S32 : ctl, Nat,  Bit, Nat, Bit, Bit, Bit, Bit -> eframe

(* Trames de supervision *)
        REJ_S10, REJ_S11, REJ_S12, REJ_S13,
        REJ_S30, REJ_S31, REJ_S32, REJ_S33,
        RR_S10, RR_S11, RR_S12, RR_S13, RR_S30, RR_S31, RR_S32, RR_S33, RR_S41,
        RNR_S10, RNR_S11, RNR_S12, RNR_S13,
        RNR_S30, RNR_S31, RNR_S32, RNR_S33 : Nat -> eframe
        RR_S1F : Nat, Bit -> eframe

(* Trames d'information *)
        I_S10, I_S11, I_S12, I_S30, I_S31, I_S32,
        I_S40, I_S41 : Nat, Nat -> eframe

    eqns
        forall v_s, v_r, n_s, n_r : Nat, final : Bit, p : packet,
            c : ctl, c_r, w, x, y, z : Bit
        ofsort eframe

(* Trames "unnumbered" *)

        DISC_S10 = m(make_disc(flagging, b, ctl_uframe(0), fc, flagging), noinfor,
            correct, notless, noabort);
        DISC_S11 = m(make_disc(flagging, b, ctl_uframe(1), fc, flagging), noinfor,
            correct, notless, noabort);
        DISC_S30 = m(make_disc(flagging, a, ctl_uframe(0), fc, flagging), noinfor,
            correct, notless, noabort);
        DISC_S31 = m(make_disc(flagging, a, ctl_uframe(1), fc, flagging), noinfor,
            correct, notless, noabort);
        DISC_S40 = m(make_disc(flagging, corrupt_adr(a), ctl_uframe(0), fc, flagging),
            noinfor, correct, notless, noabort);

        SABM_S10 = m(make_sabm(flagging, b, ctl_uframe(0), fc, flagging), noinfor,
            correct, notless, noabort);
        SABM_S11 = m(make_sabm(flagging, b, ctl_uframe(1), fc, flagging), noinfor,
            correct, notless, noabort);
        SABM_S30 = m(make_sabm(flagging, a, ctl_uframe(0), fc, flagging), noinfor,
            correct, notless, noabort);
        SABM_S31 = m(make_sabm(flagging, a, ctl_uframe(1), fc, flagging), noinfor,
            correct, notless, noabort);
        SABM_S40 = m(make_sabm(flagging, corrupt_adr(a), ctl_uframe(0), fc, flagging),
            noinfor, correct, notless, noabort);
        SABM_S41 = m(make_sabm(flagging, corrupt_adr(a), ctl_uframe(1), fc, flagging),
            noinfor, correct, notless, noabort);

        UA_S10 = m(make_ua(flagging, b, ctl_uframe(0), fc, flagging), noinfor,
            correct, notless, noabort);
        UA_S11 = m(make_ua(flagging, b, ctl_uframe(1), fc, flagging), noinfor,
            correct, notless, noabort);
        UA_S30 = m(make_ua(flagging, a, ctl_uframe(0), fc, flagging), noinfor,
            correct, notless, noabort);
        UA_S31 = m(make_ua(flagging, a, ctl_uframe(1), fc, flagging), noinfor,
            correct, notless, noabort);

        DM_S10 = m(make_dm(flagging, b, ctl_uframe(0), fc, flagging), noinfor,
            correct, notless, noabort);
        DM_S11 = m(make_dm(flagging, b, ctl_uframe(1), fc, flagging), noinfor,
            correct, notless, noabort);
        DM_S30 = m(make_dm(flagging, a, ctl_uframe(0), fc, flagging), noinfor,
            correct, notless, noabort);
        DM_S31 = m(make_dm(flagging, a, ctl_uframe(1), fc, flagging), noinfor,
            correct, notless, noabort);

        UA_S1F(final) = m(make_ua(flagging, b, ctl_uframe(final), fc, flagging), noinfor,
            correct, notless, noabort);
        DM_S1F(final) = m(make_dm(flagging, b, ctl_uframe(final), fc, flagging), noinfor,
            correct, notless, noabort);

(*
    Il y a un bit non-identifie dans les 5 derniers du make_reason
*)
```

```
        FRMR_S10(c, v_s, c_r, v_r, w, x, y, z) = m(make_frmr(flagging, b, ctl_uframe(0),
            make_reason(c, 0, v_s, c_r, v_r, w, x, y, z, 0), fc, flagging),
            noinfor, correct, notless, noabort);
        FRMR_S11(c, v_s, c_r, v_r, w, x, y, z) = m(make_frmr(flagging, b, ctl_uframe(1),
            make_reason(c, 0, v_s, c_r, v_r, w, x, y, z, 0), fc, flagging),
            noinfor, correct, notless, noabort);
        FRMR_S30(c, v_s, c_r, v_r, w, x, y, z) = m(make_frmr(flagging, a, ctl_uframe(0),
            make_reason(c, 0, v_s, c_r, v_r, w, x, y, z, 0), fc, flagging),
            noinfor, correct, notless, noabort);
        FRMR_S31(c, v_s, c_r, v_r, w, x, y, z) = m(make_frmr(flagging, a, ctl_uframe(1),
            make_reason(c, 0, v_s, c_r, v_r, w, x, y, z, 0), fc, flagging),
            noinfor, correct, notless, noabort);
        FRMR_S32(c, v_s, c_r, v_r, w, x, y, z) = m(make_frmr(flagging, a, ctl_uframe(0),
            make_reason(c, 0, v_s, c_r, v_r, w, x, y, z, 0), fc, flagging),
            noinfor, correct, notless, noabort);

(* Trames de supervision *)

        RR_S10(n_r) = m(make_rr(flagging, b, ctl_sframe(0, n_r), fc, flagging), noinfor,
            correct, notless, noabort); (* Command *)
        RR_S11(n_r) = m(make_rr(flagging, b, ctl_sframe(1, n_r), fc, flagging), noinfor,
            correct, notless, noabort); (* Command *)
        RR_S12(n_r) = m(make_rr(flagging, b, ctl_sframe(0, n_r), fc, flagging), noinfor,
            correct, notless, noabort); (* Response *)
        RR_S13(n_r) = m(make_rr(flagging, b, ctl_sframe(1, n_r), fc, flagging), noinfor,
            correct, notless, noabort); (* Response *)
        RR_S30(n_r) = m(make_rr(flagging, a, ctl_sframe(0, n_r), fc, flagging), noinfor,
            correct, notless, noabort); (* Command *)
        RR_S31(n_r) = m(make_rr(flagging, a, ctl_sframe(1, n_r), fc, flagging), noinfor,
            correct, notless, noabort); (* Command *)
        RR_S32(n_r) = m(make_rr(flagging, a, ctl_sframe(0, n_r), fc, flagging), noinfor,
            correct, notless, noabort); (* Response *)
        RR_S33(n_r) = m(make_rr(flagging, a, ctl_sframe(1, n_r), fc, flagging), noinfor,
            correct, notless, noabort); (* Response *)
        RR_S41(n_r) = m(make_rr(flagging, corrupt_adr(a), ctl_sframe(1, n_r), fc,
            flagging), noinfor, correct, notless, noabort);

        RNR_S10(n_r) = m(make_rnr(flagging, b, ctl_sframe(0, n_r), fc, flagging), noinfor,
            correct, notless, noabort); (* Command *)
        RNR_S11(n_r) = m(make_rnr(flagging, b, ctl_sframe(1, n_r), fc, flagging), noinfor,
            correct, notless, noabort); (* Command *)
        RNR_S12(n_r) = m(make_rnr(flagging, b, ctl_sframe(0, n_r), fc, flagging), noinfor,
            correct, notless, noabort); (* Response *)
        RNR_S13(n_r) = m(make_rnr(flagging, b, ctl_sframe(1, n_r), fc, flagging), noinfor,
            correct, notless, noabort); (* Response *)
        RNR_S30(n_r) = m(make_rnr(flagging, a, ctl_sframe(0, n_r), fc, flagging), noinfor,
            correct, notless, noabort); (* Command *)
        RNR_S31(n_r) = m(make_rnr(flagging, a, ctl_sframe(1, n_r), fc, flagging), noinfor,
            correct, notless, noabort); (* Command *)
        RNR_S32(n_r) = m(make_rnr(flagging, a, ctl_sframe(0, n_r), fc, flagging), noinfor,
            correct, notless, noabort); (* Response *)
        RNR_S33(n_r) = m(make_rnr(flagging, a, ctl_sframe(1, n_r), fc, flagging), noinfor,
            correct, notless, noabort); (* Response *)
        REJ_S10(n_r) = m(make_rej(flagging, b, ctl_sframe(0, n_r), fc, flagging), noinfor,
            correct, notless, noabort); (* Command *)
        REJ_S11(n_r) = m(make_rej(flagging, b, ctl_sframe(1, n_r), fc, flagging), noinfor,
            correct, notless, noabort); (* Command *)
        REJ_S12(n_r) = m(make_rej(flagging, b, ctl_sframe(0, n_r), fc, flagging), noinfor,
            correct, notless, noabort); (* Response *)
        REJ_S13(n_r) = m(make_rej(flagging, b, ctl_sframe(1, n_r), fc, flagging), noinfor,
            correct, notless, noabort); (* Response *)
        REJ_S30(n_r) = m(make_rej(flagging, a, ctl_sframe(0, n_r), fc, flagging), noinfor,
            correct, notless, noabort); (* Command *)
        REJ_S31(n_r) = m(make_rej(flagging, a, ctl_sframe(1, n_r), fc, flagging), noinfor,
            correct, notless, noabort); (* Command *)
        REJ_S32(n_r) = m(make_rej(flagging, a, ctl_sframe(0, n_r), fc, flagging), noinfor,
            correct, notless, noabort); (* Response *)
        REJ_S33(n_r) = m(make_rej(flagging, a, ctl_sframe(1, n_r), fc, flagging), noinfor,
            correct, notless, noabort); (* Response *)

        RR_S1F(n_r, final) = m(make_rr(flagging, b, ctl_sframe(final, n_r), fc, flagging),
            noinfor, correct, notless, noabort);

(* Trames d'information *)

        I_S10(n_s, n_r) = m(make_iframe(flagging, b, ctl_iframe(n_s, 0, n_r), p, fc,
            flagging), infor, correct, notless, noabort);
        I_S11(n_s, n_r) = m(make_iframe(flagging, b, ctl_iframe(n_s, 1, n_r), p, fc,
            flagging), infor, correct, notless, noabort);
        I_S12(n_s, n_r) = m(make_iframe(flagging, b, ctl_iframe(n_s, 0, n_r), p, fc,
            flagging), infor, correct, notless, noabort);
        I_S30(n_s, n_r) = m(make_iframe(flagging, a, ctl_iframe(n_s, 1, n_r), p, fc,
            flagging), infor, correct, notless, noabort);
        I_S31(n_s, n_r) = m(make_iframe(flagging, a, ctl_iframe(n_s, 0, n_r), p, fc,
            flagging), infor, correct, notless, noabort);
        I_S32(n_s, n_r) = m(make_iframe(flagging, a, ctl_iframe(n_s, 0, n_r), p, fc,
            flagging), noinfor, correct, notless, noabort); (* Pas de champ info *)
        I_S40(n_s, n_r) = m(make_iframe(flagging, a, ctl_iframe(n_s, 0, n_r), p, fc,
            flagging), infor, correct, notless, noabort);
        I_S41(n_s, n_r) = m(make_iframe(flagging, corrupt_adr(a),
            ctl_iframe(n_s, 1, n_r), p, fc, flagging), infor, correct, notless, noabort);

endtype (* Constraint *)
(* ------------------------------------------------------------------------
module Constraint is NaturalNumber, Boolean, Extframetype, Frametype, Packet,
    Infor, Corrlength, Lesslength, Abortf, Mode
    type constr is
    endtype

/* Trames "unnumbered */
    function DISC_S10 : eframe is
        m(make_disc(flagging, b, ctl_uframe(0), fc, flagging), noinfor,
            correct, notless, noabort)
    endfunc
    function DISC_S11 : eframe is
        m(make_disc(flagging, b, ctl_uframe(1), fc, flagging), noinfor,
            correct, notless, noabort)
```

```
endfunc
function DISC_S30 : eframe is
    m(make_disc(flagging, a, ctl_uframe(0), fc, flagging), noinfor,
        correct, notless, noabort)
endfunc
function DISC_S31 : eframe is
    m(make_disc(flagging, a, ctl_uframe(1), fc, flagging), noinfor,
        correct, notless, noabort)
endfunc
function DISC_S40 : eframe is
    m(make_disc(flagging, corrupt_adr(a), ctl_uframe(0), fc, flagging),
        noinfor, correct, notless, noabort)
endfunc
function SABM_S10 : eframe is
    m(make_sabm(flagging, b, ctl_uframe(0), fc, flagging), noinfor,
        correct, notless, noabort)
endfunc
function SABM_S11 : eframe is
    m(make_sabm(flagging, b, ctl_uframe(1), fc, flagging), noinfor,
        correct, notless, noabort)
endfunc
function SABM_S30 : eframe is
    m(make_sabm(flagging, a, ctl_uframe(0), fc, flagging), noinfor,
        correct, notless, noabort)
endfunc
function SABM_S31 : eframe is
    m(make_sabm(flagging, a, ctl_uframe(1), fc, flagging), noinfor,
        correct, notless, noabort)
endfunc
function SABM_S40 : eframe is
    m(make_sabm(flagging, corrupt_adr(a), ctl_uframe(0), fc, flagging),
        noinfor, correct, notless, noabort)
endfunc
function SABM_S41 : eframe is
    m(make_sabm(flagging, corrupt_adr(a), ctl_uframe(1), fc, flagging),
        noinfor, correct, notless, noabort)
endfunc

function UA_S10 : eframe is
    m(make_ua(flagging, b, ctl_uframe(0), fc, flagging), noinfor,
        correct, notless, noabort)
endfunc
function UA_S11 : eframe is
    m(make_ua(flagging, b, ctl_uframe(1), fc, flagging), noinfor,
        correct, notless, noabort)
endfunc
function UA_S30 : eframe is
    m(make_ua(flagging, a, ctl_uframe(0), fc, flagging), noinfor,
        correct, notless, noabort)
endfunc
function UA_S31 : eframe is
    m(make_ua(flagging, a, ctl_uframe(1), fc, flagging), noinfor,
        correct, notless, noabort)
endfunc
```

```
function DM_S10 : eframe is
    m(make_dm(flagging, b, ctl_uframe(0), fc, flagging), noinfor,
        correct, notless, noabort)
endfunc
function DM_S11 : eframe is
    m(make_dm(flagging, b, ctl_uframe(1), fc, flagging), noinfor,
        correct, notless, noabort)
endfunc
function DM_S30 : eframe is
    m(make_dm(flagging, a, ctl_uframe(0), fc, flagging), noinfor,
        correct, notless, noabort)
endfunc
function DM_S31 : eframe is
    m(make_dm(flagging, a, ctl_uframe(1), fc, flagging), noinfor,
        correct, notless, noabort)
endfunc

function UA_S1F(final:Bit) : eframe is
    m(make_ua(flagging, b, ctl_uframe(final), fc, flagging), noinfor,
        correct, notless, noabort)
endfunc

function DM_S1F(final:Bit) : eframe is
    m(make_dm(flagging, b, ctl_uframe(final), fc, flagging), noinfor,
        correct, notless, noabort)
endfunc
function UA_S1F(final:Bit) : eframe is
     m(make_ua(flagging, b, ctl_uframe(final), fc, flagging), noinfor,
        correct, notless, noabort)
endfunc
function DM_S1F(final:Bit) : eframe is
    m(make_dm(flagging, b, ctl_uframe(final), fc, flagging), noinfor,
        correct, notless, noabort)
endfunc
function  FRMR_S10(c: ctl,v_s: Nat,c_r: Bit,v_r: Nat,
            w: Bit,x: Bit,y: Bit,z: Bit) : eframe is
     m(make_frmr(flagging, b, ctl_uframe(0),
        make_reason(c, 0, v_s, c_r, v_r, w, x, y, z, 0), fc, flagging),
        noinfor, correct, notless, noabort)
endfunc
function  FRMR_S11(c: ctl,v_s: Nat,c_r: Bit,v_r: Nat,
            w: Bit,x: Bit,y: Bit,z: Bit) : eframe is
    m(make_frmr(flagging, b, ctl_uframe(1),
        make_reason(c, 0, v_s, c_r, v_r, w, x, y, z, 0), fc, flagging),
        noinfor, correct, notless, noabort)
endfunc
function  FRMR_S30(c: ctl,v_s: Nat,c_r: Bit,v_r: Nat,
            w: Bit,x: Bit,y: Bit,z: Bit) : eframe is
     m(make_frmr(flagging, a, ctl_uframe(0),
        make_reason(c, 0, v_s, c_r, v_r, w, x, y, z, 0), fc, flagging),
        noinfor, correct, notless, noabort)
endfunc
function  FRMR_S31(c: ctl,v_s: Nat,c_r: Bit,v_r: Nat,
                w: Bit,x: Bit,y: Bit,z: Bit) : eframe is
        m(make_frmr(flagging, a, ctl_uframe(1),
            make_reason(c, 0, v_s, c_r, v_r, w, x, y, z, 0), fc, flagging),
            noinfor, correct, notless, noabort)
    endfunc
    function  FRMR_S32(c: ctl,v_s: Nat,c_r: Bit,v_r: Nat,
                w: Bit,x: Bit,y: Bit,z: Bit) : eframe is
         m(make_frmr(flagging, a, ctl_uframe(0),
            make_reason(c, 0, v_s, c_r, v_r, w, x, y, z, 0), fc, flagging),
            noinfor, correct, notless, noabort)
    endfunc

/* Trames de supervision */
    function REJ_S10 (n_r: Nat) : eframe is
        m(make_rej(flagging, b, ctl_sframe(0, n_r), fc, flagging), noinfor,
            correct, notless, noabort)
    endfunc
    function REJ_S11 (n_r: Nat) : eframe is
        m(make_rej(flagging, b, ctl_sframe(1, n_r), fc, flagging), noinfor,
            correct, notless, noabort)
    endfunc
    function REJ_S12 (n_r: Nat) : eframe is
        m(make_rej(flagging, b, ctl_sframe(0, n_r), fc, flagging), noinfor,
            correct, notless, noabort)
    endfunc
    function REJ_S13 (n_r: Nat) : eframe is
        m(make_rej(flagging, b, ctl_sframe(1, n_r), fc, flagging), noinfor,
            correct, notless, noabort)
    endfunc
    function REJ_S30 (n_r: Nat) : eframe is
        m(make_rej(flagging, a, ctl_sframe(0, n_r), fc, flagging), noinfor,
            correct, notless, noabort)
    endfunc
    function REJ_S31 (n_r: Nat) : eframe is
        m(make_rej(flagging, a, ctl_sframe(1, n_r), fc, flagging), noinfor,
            correct, notless, noabort)
    endfunc
    function REJ_S32 (n_r: Nat) : eframe is
        m(make_rej(flagging, a, ctl_sframe(0, n_r), fc, flagging), noinfor,
            correct, notless, noabort))
    endfunc
    function REJ_S33 (n_r: Nat) : eframe is
        m(make_rej(flagging, a, ctl_sframe(1, n_r), fc, flagging), noinfor,
            correct, notless, noabort)
    endfunc

    function RR_S10 (n_r: Nat) : eframe is
        m(make_rr(flagging, b, ctl_sframe(0, n_r), fc, flagging), noinfor,
            correct, notless, noabort)
    endfunc
    function RR_S11 (n_r: Nat) : eframe is
        m(make_rr(flagging, b, ctl_sframe(1, n_r), fc, flagging), noinfor,
            correct, notless, noabort)
    endfunc
```

```
function RR_S12 (n_r: Nat) : eframe is
    m(make_rr(flagging, b, ctl_sframe(0, n_r), fc, flagging), noinfor,
        correct, notless, noabort)
endfunc
function RR_S13 (n_r: Nat) : eframe is
     m(make_rr(flagging, b, ctl_sframe(1, n_r), fc, flagging), noinfor,
        correct, notless, noabort)
endfunc
function RR_S30 (n_r: Nat) : eframe is
     m(make_rr(flagging, a, ctl_sframe(0, n_r), fc, flagging), noinfor,
        correct, notless, noabort)
endfunc
function RR_S31 (n_r: Nat) : eframe is
     m(make_rr(flagging, a, ctl_sframe(0, n_r), fc, flagging), noinfor,
        correct, notless, noabort)
endfunc
function RR_S32 (n_r: Nat) : eframe is
     m(make_rr(flagging, a, ctl_sframe(0, n_r), fc, flagging), noinfor,
        correct, notless, noabort)
endfunc
function RR_S33 (n_r: Nat) : eframe is
    m(make_rr(flagging, a, ctl_sframe(1, n_r), fc, flagging), noinfor,
        correct, notless, noabort)
endfunc
function RR_S41 (n_r: Nat) : eframe is
    m(make_rr(flagging, corrupt_adr(a), ctl_sframe(1, n_r), fc,
        flagging), noinfor, correct, notless, noabort)
endfunc
function RNR_S10 (n_r: Nat) : eframe is
     m(make_rnr(flagging, b, ctl_sframe(0, n_r), fc, flagging), noinfor,
        correct, notless, noabort)
endfunc
function RNR_S11 (n_r: Nat) : eframe is
    m(make_rnr(flagging, b, ctl_sframe(1, n_r), fc, flagging), noinfor,
        correct, notless, noabort)
endfunc
function RNR_S12 (n_r: Nat) : eframe is
    m(make_rnr(flagging, b, ctl_sframe(0, n_r), fc, flagging), noinfor,
        correct, notless, noabort)
endfunc
function RNR_S13 (n_r: Nat) : eframe is
    m(make_rnr(flagging, b, ctl_sframe(1, n_r), fc, flagging), noinfor,
        correct, notless, noabort)
endfunc
function RNR_S30 (n_r: Nat) : eframe is
    m(make_rnr(flagging, a, ctl_sframe(0, n_r), fc, flagging), noinfor,
        correct, notless, noabort)
endfunc
function RNR_S31 (n_r: Nat) : eframe is
    m(make_rnr(flagging, a, ctl_sframe(1, n_r), fc, flagging), noinfor,
        correct, notless, noabort)
endfunc
function RNR_S32 (n_r: Nat) : eframe is
    m(make_rnr(flagging, a, ctl_sframe(0, n_r), fc, flagging), noinfor,
            correct, notless, noabort)
    endfunc
    function RNR_S33 (n_r: Nat) : eframe is
        m(make_rnr(flagging, a, ctl_sframe(1, n_r), fc, flagging), noinfor,
            correct, notless, noabort)
    endfunc
    function RR_S1F (n_r: Nat, final: Bit) : eframe is
        m(make_rr(flagging, b, ctl_sframe(final, n_r), fc, flagging),
            noinfor, correct, notless, noabort)
    endfunc

/* Trames d'information */
    function I_S10 (n_s: Nat, n_r: Nat) : eframe is
        m(make_iframe(flagging, b, ctl_iframe(n_s, 0, n_r), p of packet, fc,
            flagging), infor, correct, notless, noabort)
    endfunc
    function I_S11 (n_s: Nat, n_r: Nat) : eframe is
        m(make_iframe(flagging, b, ctl_iframe(n_s, 1, n_r), p of packet, fc,
            flagging), infor, correct, notless, noabort)
    endfunc
    function I_S12 (n_s: Nat, n_r: Nat) : eframe is
        m(make_iframe(flagging, b, ctl_iframe(n_s, 0, n_r), p of packet, fc,
            flagging), infor, correct, notless, noabort)
    endfunc
    function I_S30 (n_s: Nat, n_r: Nat) : eframe is
        m(make_iframe(flagging, a, ctl_iframe(n_s, 1, n_r), p of packet, fc,
            flagging), infor, correct, notless, noabort)
    endfunc
    function I_S31 (n_s: Nat, n_r: Nat) : eframe is
        m(make_iframe(flagging, a, ctl_iframe(n_s, 0, n_r), p of packet, fc,
            flagging), infor, correct, notless, noabort)
    endfunc
    function I_S32 (n_s: Nat, n_r: Nat) : eframe is
        m(make_iframe(flagging, a, ctl_iframe(n_s, 0, n_r), p of packet, fc,
            flagging), noinfor, correct, notless, noabort)
    endfunc
    function I_S40 (n_s: Nat, n_r: Nat) : eframe is
        m(make_iframe(flagging, a, ctl_iframe(n_s, 0, n_r), p of packet, fc,
            flagging), infor, correct, notless, noabort)
    endfunc
    function I_S41 (n_s: Nat, n_r: Nat) : eframe is
        m(make_iframe(flagging, corrupt_adr(a), ctl_iframe(n_s, 1, n_r), p of packet, fc,
            flagging), infor, correct, notless, noabort)
    endfunc

endmod
========================================================================== *)

type    ContradictoryPredicates is Boolean, Extframetype
(* sorts   contpred *)
opns    IsUA_F1, IsDM_F1, IsDISC, IsSABMorSABME,
            NotUAorDMorDISCorSABM : eframe -> Bool
eqns    forall f : eframe
        ofsort Bool
        IsUA_F1(f) = correct(f) and (is_ua(get_f(f)) and (pf(f) eq 1 of Bit));
        IsDM_F1(f) = correct(f) and (is_dm(get_f(f)) and (pf(f) eq 1 of Bit));
        IsDISC(f) =  correct(f) and is_disc(get_f(f));
        IsSABMorSABME(f) = correct(f) and (is_sabm(get_f(f)) or
            is_sabme(get_f(f)));
        NotUAorDMorDISCorSABM(f) = not(IsUA_F1(f) or IsDM_F1(f) or IsDISC(f) or
            IsSABMorSABME(f))
endtype (* ContradictoryPredicates *)
(* ---------------------------------------------------------------------------
module    ContradictoryPredicates is Boolean, Extframetype
type   contpred
endtype

    function IsUA_F1 (F: eframe) : Bool is
        correct(F) and (is_ua(get_f(F)) and (pf(F) eq 1 of Bit))
    endfunc
    function IsDM_F1 (F: eframe) : Bool is
        correct(F) and (is_dm(get_f(F)) and (pf(F) eq 1 of Bit))
    endfunc
    function IsDISC (F: eframe) : Bool is
        correct(F) and is_disc(get_f(F))
    endfunc
    function IsSABMorSABME (F: eframe) : Bool is
        correct(F) and (is_sabm(get_f(F)) or is_sabme(get_f(F)))
    endfunc
    function NotUAorDMorDISCorSABM (F: eframe) : Bool is
        not(IsUA_F1(F) or IsDM_F1(F) or IsDISC(F) or  IsSABMorSABME(F))
    endfunc
endmod
========================================================================== *)

behaviour
    hide phl, dl, t in
    (
        (
            (
                lapb[dl, phl, t](adr(dte), adr(dce), n2, k, tmode,
                    resolvecollis, senddmoption)
                |[dl]|
                Value_Provider[dl]
            )
            |[t]|
            Timer[t]
        )
        |[phl]|
        Medium[phl, L, TM]
    )

    where

        process Global_Variables[V_S, V_R, PF_bit](vs, vr: Nat,
            pf: Bit) : noexit :=
        (
            V_S ! Read ! vs;
            Global_Variables[V_S, V_R, PF_bit](vs, vr, pf)
        []
            V_S ! Write ? s: Nat;
            Global_Variables[V_S, V_R, PF_bit](s, vr, pf)
        []
            V_R ! Read ! vr;
            Global_Variables[V_S, V_R, PF_bit](vs, vr, pf)
        []
            V_R ! Write ? s: Nat;
            Global_Variables[V_S, V_R, PF_bit](vs, s, pf)
        []
            PF_bit ! Read ! pf;
            Global_Variables[V_S, V_R, PF_bit](vs, vr, pf)
        []
            PF_bit ! Write ? b: Bit;
            Global_Variables[V_S, V_R, PF_bit](vs, vr, b)
        )
        endproc (* Global_Variables *)

        process Value_Provider[dl] : noexit :=
        (*
            Value_Provider permet de fournir et accepter des valeurs a la
            porte dl qui ne participe pas aux interactions des processus
            referant aux cas de tests.
        *)
                dl ! dl_est_req; Value_Provider[dl]
            []
                dl ! dl_est_ind; Value_Provider[dl]
            []
                dl ! dl_est_cnf; Value_Provider[dl]
            []
                dl ! dl_dis_req; Value_Provider[dl]
            []
                dl ! dl_dis_ind; Value_Provider[dl]
            []
                dl ! dl_dis_cnf; Value_Provider[dl]
            []
                dl ! dl_data_req(Bit(0)); Value_Provider[dl]
            []
                dl ! dl_data_ind(Bit(0)); Value_Provider[dl]
        endproc (* Value_Provider *)

        process Timer[tm] : noexit :=
            tm ! start;
            (
                tm ! expired;
                Timer[tm] (* ? *)
            []
                tm ! reset;
                Timer[tm]
            )
        endproc (* Timer *)
```

```
        process Medium[phl, L, TM] : noexit :=
            L ! Input ? f: eframe;
            phl ! Out ! f;
            Medium[phl, L, TM]
        []
            L ! Out ? f: eframe;
            phl ! Input ! f;
            Medium[phl, L, TM]
        []
            TM; (* Timeout *)
            Medium[phl, L, TM]
        endproc (* Medium *)

(******************************************************************************

                L A P B  S P E C I F I C A T I O N

 ******************************************************************************)

process lapb[dl, phl, t](from, to: adress, n2, k: Nat,
    tmode: mmode, resolvecollis: Nat, senddmoption: Bool) : noexit :=

(* [k lt k(tmode)] -> *)
    (
        (
            connectionphase[dl, phl, t](from, to, n2, k, tmode, resolvecollis,
                senddmoption) >>
            accept ok: Bool, donotwaitua: Bool in
        (
                [ok] ->
(* ---------------------------------------------------------------------------
                if ok then
=========================================================================== *)
                (
                    hide pp in
                (
                        (
                            dataphase[dl,phl,pp,t](donotwaitua, from, to, n2,
                                k, tmode, empty of pqueue, empty of pqueue,
                                empty of pacqueue, resolvecollis)
                    [>
                        terminationphase[dl, phl, pp, t](from, to, n2, k,
                                tmode, resolvecollis)
                    )
                    |[pp]|
                    goterm[pp]
                )
                )
        []
            [not(ok)] ->
(* ---------------------------------------------------------------------------
                else
=========================================================================== *)
                exit
```

```
(* -------------------------------------------------------------------------
                endif
======================================================================== *)

        )
        ) >>
    lapb[dl, phl, t](from, to, n2, k, tmode, resolvecollis, senddmoption)
    )

where
    process connectionphase[dl, phl, t] (from, to: adress,
        n2, k: Nat, tmode: mmode, resolvecollis: Nat,
        senddmoption: Bool) : exit(Bool, Bool)  :=
    (
        disconnectedphase[phl](from, to, n2, k, tmode) >>
        connectionphase[dl, phl, t](from, to, n2, k, tmode, resolvecollis,
            senddmoption)
    )
    []
        calling[dl, phl, t](from, to, n2, k, tmode, resolvecollis,
            senddmoption)
    []
        called[dl, phl, t](from, to, n2, k, tmode, resolvecollis)
    []
    (
       receiveincorrect[phl] >>
       connectionphase[dl, phl, t](from, to, n2, k, tmode, resolvecollis,
           senddmoption)
    )

    where
        process called[dl, phl, t](from, to: adress, n2, k: Nat,
            tmode: mmode, resolvecollis: Nat) : exit(Bool, Bool) :=

            phl ! Input ? sframe: eframe[(((is_sabm(get_f(sframe))) and
                (eq(tmode, basic))) or ((is_sabme(get_f(sframe)))
                and (eq(tmode, extended))) ) and (correct(sframe))];
            (
                i; phl ! Out ! m(make_ua(flagging, from,
                    ctl_uframe(pf(((sframe)))), fc, flagging), noinfor,
                    correct, notless, noabort);
                exit(true of Bool, false)
            []
                i; phl ! Out ! m(make_dm(flagging, from,
                    ctl_uframe(pf(sframe)), fc, flagging), noinfor, correct,
                    notless, noabort);
                exit(false, false)
            )
        []
            phl ! Input ? sframe: eframe [(correct(sframe)) and
                ((is_dm(get_f(sframe))) and (pf(((sframe))) eq 0))];
           (
              (* request for link set up accepted *)
              i; calloption1[dl, phl, t](m(make_sabm(flagging, to,
                    ctl_uframe(1), fc, flagging), noinfor, correct,
                    notless, noabort), from, to, n2, k, tmode,
                    resolvecollis)
           []
               (* requeset for link set up refused  *)
              i; phl ! Out ! m(make_disc(flagging, to, ctl_uframe(pf(sframe)),
                 fc, flagging), noinfor, correct, notless, noabort);
              exit(false, false)
           )

        endproc (* called *)

        process disconnectedphase[phl](from: adress, to: adress, n2: Nat,
            k: Nat, tmode: mmode) : exit :=
            (
                (
                    phl ! Input ? sframe: eframe [((correct(sframe)) and
                        ((pf(sframe) eq 1 of Bit) and (eq(adrs(sframe),
                        from)))) or (is_disc(get_f(sframe)))];
                    exit(pf(sframe))
                )
                |[phl]|
                (
                    phl ! Input ? sframe: eframe [not(((eq(tmode, basic)) and
                        (is_sabm(get_f(sframe)))) or ((eq(tmode, extended))
                        and (is_sabme(get_f(sframe)))))];
                    exit(pf(sframe))
                ) >>
                accept b: Bit in
                phl ! Out ! m(make_dm(flagging, from, ctl_uframe(b), fc,
                    flagging), noinfor, correct, notless, noabort);
                exit
            )
        []
            (
                (
                    phl ! Input ? sframe: eframe [not(((correct(sframe)) and
                        ((pf(sframe) eq 1 of Bit) and
                        (eq(adrs(sframe), from)))) or (is_disc(get_f(sframe))))
                        and not ((is_dm(get_f(sframe))) and
                        (pf(sframe) eq 0 of Bit))];
                    exit
                )
                |[phl]|
                (
                    phl ! Input ? sframe: eframe [not(((eq(tmode,basic)) and
                        (is_sabm(get_f(sframe)))) or ((eq(tmode, extended))
                        and (is_sabme(get_f(sframe)))))];
                    exit
                )
            )

        endproc (* disconnectedphase *)
```

```
        process calling[dl, phl, t](from, to: adress, n2, k: Nat,
            tmode: mmode, resolvecollis: Nat, senddmoption: Bool) :
            exit(Bool, Bool) :=

            dl ? p: primitive [is_dl_est_req(p)];
            (
                (
                    [eq(tmode, basic)] ->
(* --------------------------------------------------------------------------
                case tmode in
                  basic ->
========================================================================== *)
                    (
                        calloption1[dl, phl, t](m(make_sabm(flagging, to,
                            ctl_uframe(1), fc, flagging), noinfor, correct,
                            notless, noabort), from, to, n2, k, tmode,
                            resolvecollis)
                    )
            []
                [eq(tmode, extended)] ->
(* --------------------------------------------------------------------------
                | extended ->
========================================================================== *)
                    (
                        calloption1[dl, phl, t](m(make_sabme(flagging, to,
                            ctl_uframe(1), fc, flagging), noinfor, correct,
                            notless, noabort), from, to, n2, k, tmode,
                            resolvecollis)
                    )
(* --------------------------------------------------------------------------
                endcase
========================================================================== *)
            )
        []
                calloption2[dl, phl, t](from, to, n2, k, tmode, resolvecollis)
        []
            calloption3[dl, phl, t](from, to, n2, k, tmode, resolvecollis,
                    senddmoption)
        )
        endproc (* calling *)

    process calloption2[dl, phl, t] (from, to: adress, n2, k: Nat,
            tmode: mmode,resolvecollis: Nat) : exit(Bool, Bool) :=

            calloption1[dl, phl, t](m(make_disc(flagging, to, ctl_uframe(1),
                fc, flagging), noinfor, correct, notless, noabort), from, to,
                n2, k, tmode, resolvecollis) >>
        accept ok: Bool, donotwait: Bool in
            (* ok = true of Bool si DTE est pret a etablir la connexion *)
        (
                [ok] ->
            (
                    [eq(tmode, basic)] ->
                (
                        calloption1[dl, phl, t](m(make_sabm(flagging, to,
                            ctl_uframe(1), fc, flagging), noinfor, correct,
                            notless, noabort), from, to, n2, k, tmode,
                            resolvecollis)
                    )
            []
                [eq(tmode,extended)] ->
                (
                        calloption1[dl, phl, t](m(make_sabme(flagging, to,
                            ctl_uframe(1), fc, flagging), noinfor, correct,
                            notless, noabort), from, to, n2, k, tmode,
                            resolvecollis)
                    )
                )
        [] (* La procedure de connexion n'a pas ete initiee par le DTE *)
            [not(ok)] ->
            (
                    exit(false, false)
                )
        )
        endproc (* calloption2 *)

        process calloption3[dl, phl, t](from, to: adress, n2, k: Nat,
            tmode: mmode, resolvecollis: Nat, senddmoption: Bool) :
            exit(Bool, Bool) :=

        [senddmoption] ->
        (
            hide pp, p in
            senddm[phl, pp, p, t](from, to, 0 of Nat, n2)
            |[phl, pp, p]|
            waitrespdm[dl, phl, pp, p, t](from, to, n2, k, tmode,
                    resolvecollis)
        )

         where
            process senddm[phl, pp, p, t](from, to: adress, init, n2: Nat) :
                exit(Bool, Bool) :=
        (
                [init lt n2] ->
                (
                    p ! Succ(0) of Nat;
                    phl ! Out ! m(make_dm(flagging, to, ctl_uframe(0), fc,
                        flagging), noinfor, correct, notless, noabort);
                    t ! start;
                    synchrespdm[phl, pp, t, p]
                 [>
                     (
                         t ! expired;
                         exit >>
                         senddm[phl, pp, p, t](from, to, Succ(init), n2)
                 )
                 )
        []
```

```
        [init eq n2] ->
            (
                p ! 0 of Bit;
                exit(false, false)
        )
        )

    where
        process synchrespdm[phl, pp, t, p] : exit(Bool, Bool) :=
        (
                pp ! 0 of Nat;
                t ! reset;
                exit(any Bool, any Bool)
        )
        []
        (
                pp ! Succ(0) of Nat;
                (
                    exit(any Bool, any Bool)
            []
                wwaituadm[phl]
            )
        )
        []
        (
                phl ! Input ? fram: eframe;
                synchrespdm[phl, pp, t, p]
        )
        endproc (* synchrespdm *)
        endproc (* senddm *)

    process waitrespdm[dl, phl, pp, p, t](from, to: adress, n2, k :
            Nat, tmode: mmode, resolvecollis: Nat) : exit(Bool, Bool) :=
    (
            p ! Succ(0) of Nat;
            phl ! Input ? f: eframe;
            waitingresponsedm[dl, phl, pp, p, t](from, to, n2, k, tmode,
                resolvecollis)
    )
        [>
            p ! 0 of Bit;
            exit(any Bool, any Bool)
    where
        process waitingresponsedm[dl, phl, pp, p, t](from, to: adress,
                n2, k: Nat, tmode: mmode, resolvecollis: Nat) :
            exit(Bool, Bool) :=

                notsabmdiscdm[phl, pp, p]
            |[phl, pp, p]|
            isdm[dl, phl, pp, p, t](from, to, n2, k, tmode,
                    resolvecollis)
            |[phl, pp, p]|
            issabmofdm[dl, phl, pp, p](from, to, n2, k, tmode)
            |[phl, pp, p]|
            isdiscofdm[dl, phl, pp, p](from, to, n2, k, tmode)
        endproc (* waitingresponsedm *)
    endproc  (* waitrespdm *)
    endproc (* calloption3 *)

    process notsabmdiscdm[phl, pp, p] : exit(Bool,Bool) :=
    (
        (
            phl ! Input ? sframe: eframe [((correct(sframe) and (is_sabm
                (get_f(sframe))))) or (((is_disc(get_f(sframe))) or
                is_dm(get_f(sframe))))];
            pp ? n: Nat;
            (
                exit(any Bool,any Bool)
            []
                wwaituadm[phl])
            )
        []
        (
                phl ! Input ? sframe: eframe [not(((correct(sframe) and
                    (is_sabm(get_f(sframe))))) or
                    (((is_disc(get_f(sframe))) or
                    is_dm(get_f(sframe)))))];
                notsabmdiscdm[phl,pp,p]
        )
    )
    [>
    (
        p ! Succ(0) of Nat;
        phl ! Input ? f: eframe;
        notsabmdiscdm[phl,pp,p]
    )
    endproc (* notsabmdmf1uaf1disc *)

    process isdiscofdm[dl,phl,pp,p](from, to: adress, n2, k: Nat,
        tmode: mmode) : exit(Bool, Bool) :=
    (
        (
            phl ! Input ? f: eframe[not(correct(f) and
                is_disc(get_f(f)))];
            (
                isdiscofdm[dl, phl, pp, p](from, to, n2, k, tmode)
            []
                pp ? n: Nat;
                (
                    exit(any Bool,any Bool)
                []
                    wwaituadm[phl]
                )
            )
        )
    []
        (
            phl ! Input ? f: eframe[correct(f) and is_disc(get_f(f))];
            pp ! 0 of Nat;
            exit(false,false)
        )
    )
    [>
    (
        p ! Succ(0) of Nat;
        phl ! Input ? f: eframe;
        isdiscofdm[dl,phl,pp,p](from,to,n2,k,tmode)
    )
    endproc (* isdiscofdm *)

    process issabmofdm[dl,phl,pp,p](from, to: adress, n2, k: Nat,
        tmode: mmode) : exit(Bool, Bool) :=
    (
        (
            phl ! Input ? f: eframe[not(correct(f) and (is_sabm(get_f(f))
                or is_sabme(get_f(f))))];
            (
                issabmofdm[dl, phl, pp, p](from, to, n2, k, tmode)
            []
                pp ? n: Nat;
                (
                    exit(any Bool, any Bool)
                []
                    wwaituadm[phl]
                )
            )
        )
    []
        (
            phl ! Input ? f: eframe[correct(f) and (is_sabm(get_f(f)) or
                is_sabme(get_f(f)))];
            pp ! 0 of Nat;
        exit(true of Bool, false)
        )
    )
    [>
    (
        p ! Succ(0) of Nat;
        phl ! Input ? f: eframe;
        issabmofdm[dl, phl, pp, p](from, to, n2, k, tmode)
    )
    endproc (* issabm *)

    process isdm[dl,phl,pp,p,t](from, to: adress, n2, k: Nat,
        tmode: mmode, resolvecollis: Nat) : exit(Bool, Bool) :=
        (
            (
                phl ! Input ? f: eframe[not(correct(f) and (is_dm(get_f(f))))];
                (
                    isdm[dl, phl, pp, p, t](from, to, n2, k, tmode,
                        resolvecollis)
                []
                pp ? n: Nat;
                exit(any Bool, any Bool)
            )
        )
    []
        (
            phl ! Input ? f: eframe[correct(f) and (is_dm(get_f(f))
                and (pf(f) eq 1 of Bit))];
            pp ! 0 of Nat;
            exit(false, false)
    )
[]
    (
        phl ! Input ? f: eframe[correct(f) and (is_dm(get_f(f)) and
            (pf(f) eq 0 of Bit))];
            pp ! Succ(0) of Nat;
        (
            [eq(tmode, basic)] ->
            (
                calloption1[dl, phl, t](m(make_sabm(flagging, to,
                    ctl_uframe(1), fc, flagging), noinfor, correct,
                    notless, noabort), from, to, n2, k, tmode,
                    resolvecollis))
        []
            [eq(tmode,extended)] ->
            (
                calloption1[dl, phl, t](m(make_sabme(flagging, to,
                    ctl_uframe(1), fc, flagging), noinfor, correct,
                    notless, noabort), from, to, n2, k, tmode,
                    resolvecollis))
        )
    )
)
[>
(
    p ! Succ(0) of Nat;
    phl ! Input ? f: eframe;
    isdm[dl, phl, pp, p, t](from, to, n2, k, tmode, resolvecollis)
)
endproc (* isdm *)

process sacceptedsabm[med] : exit(eframe) :=
(* accept frame either SABM, DISC, UA or DM *)
    med ? sframe: eframe [(is_sabm(get_f(sframe))) or ((is_disc
        (get_f(sframe))) or ((is_ua(get_f(sframe))) or
        (is_dm(get_f(sframe)))))];
    exit((sframe))
endproc  (* sacceptedsabm *)


process receiveincorrect[phl] : exit :=
    phl ! Input ? f: eframe[not(correct(f))];
    exit
endproc (* receiveincorrect *)
endproc (* connection phase *)

process goterm[pp] : exit :=
(
    pp ! termine;
pp ! causetermine;
exit
)
[]
    exit
endproc (* goterm *)

process terminationphase[dl, phl, pp, t](from, to: adress, n2,
    k: Nat, tmode: mmode, resolvecollis: Nat) : exit :=
(
    dl ! dl_dis_req;       (* user entity iniates disconnection *)
    calloption1[dl, phl, t](m(make_disc(flagging, to, ctl_uframe(1), fc,
        flagging), noinfor, correct, notless, noabort), from, to, n2, k,
         tmode, resolvecollis) >>
    accept k: Bool, donotwait: Bool in
    dl ! dl_dis_cnf; exit
)
[]
(
    phl ! Input ? sframe: eframe[(correct(sframe)) and
        (is_disc(get_f(sframe)))];
    dl ! dl_dis_ind;
    phl ! Out ! m(make_ua(flagging, from, ctl_uframe(pf(sframe)), fc,
        flagging), noinfor, correct, notless, noabort);
    exit
)
[]
    pp ! causetermine;
    dl ! dl_dis_ind;
    exit
endproc (* terminationphase *)

process wwaituadm[phl] : exit(Bool, Bool) :=
    phl ! Input ? fram: eframe;
    (
        exit(any Bool, any Bool)
    []
        wwaituadm[phl]
    )
endproc (* wwaituadm *)

process wwaitua[phl] : exit(Bool, Bool, eframe) :=
    phl ! Input ? fram: eframe;
    (
        exit(any Bool, any Bool, m(make_ua(flagging, a, ctl_uframe(1), fc,
            flagging), noinfor, correct, notless, noabort))
    []
        wwaitua[phl]
    )
```

```
endproc (* wwaitua *)

process notsabmdiscuaf1dmf1[phl, pp, p] : exit(Bool, Bool, eframe) :=
(
    (
        (
            phl ! Input ? sframe: eframe[(IsSABMorSABME(sframe) or
                IsUA_F1(sframe)) or (IsDM_F1(sframe) or IsDISC(sframe))];
            (
                (
                    phl ! Input ? f: eframe;
                    pp ? n: Nat;
                    exit(any Bool, any Bool, any eframe)
                )
            []
                (
                    pp ? n: Nat;
                    exit(any Bool, any Bool, any eframe)
                )
            )
        )
    []
        (
            phl ! Input ? sframe: eframe[NotUAorDMorDISCorSABM(sframe)];
            notsabmdiscuaf1dmf1[phl, pp, p]
        )
    )
    [>
    (
        p ! Succ(Succ(0)) of Nat;
        phl ! Input ? f: eframe;
        (
            exit(any Bool, any Bool, any eframe)
       []
            wwaitua[phl]
        )
    )
)
[>
(
    p ! Succ(0) of Nat;
    phl ! Input ? f: eframe;
    notsabmdiscuaf1dmf1[phl, pp, p]
)
endproc (* notsabmdmf1uaf1disc *)

process isuaf1[phl, pp, p] : exit(Bool, Bool, eframe) :=
(
    (
        (
            (
                phl ! Input ? f: eframe[not(IsUA_F1(f))];
                (
                    isuaf1[phl,pp,p]
                []
                    pp ? n: Nat;
                    exit(any Bool, any Bool, any eframe)
                )
            )
        )
    []
        (
            phl ! Input ? f: eframe[IsUA_F1(f)];
            pp ! 0 of Nat;
            exit(true of Bool, false, f)
        )
    )
    [>
    (
        p ! Succ(Succ(0)) of Nat;
        phl ! Input ? f: eframe;
        (
            exit(any Bool, any Bool, any eframe)
        []
            wwaitua[phl]
        )
    )
)
[>
(
    p ! Succ(0) of Nat;
    phl ! Input ? f: eframe;
    isuaf1[phl,pp,p]
)
endproc (* isuaf1 *)

process isdmf1[phl, pp, p](fr: eframe) : exit(Bool, Bool, eframe) :=
(
    (
        (
            (
                phl ! Input ? f: eframe[not(IsDM_F1(f))];
                (
                    isdmf1[phl, pp, p](fr)
                []
                    pp ? n: Nat;
                    exit(any Bool, any Bool, any eframe)
                )
            )
        )
    []
        (
            phl ! Input ? f: eframe[IsDM_F1(f)];
            pp ! 0 of Nat;
            exit(resp(fr), false, f)
        )
    )
    [>
```

```
        (
            p ! Succ(Succ(0)) of Nat;
            phl ! Input ? f: eframe;
            (
                exit(any Bool, any Bool, any eframe)
            []
                wwaitua[phl]
            )
        )
    )
    [>
    (
        p ! Succ(0) of Nat;
        phl ! Input ? f: eframe;
        isdmf1[phl,pp,p](fr)
    )
    endproc (* isdmf1 *)

    process issabm[dl, phl, pp, p, t](fr: eframe, from, to: adress, n2, k :
        Nat, tmode: mmode, resolvecollis: Nat) : exit(Bool, Bool, eframe) :=
(
        (
            (
                (
                    (
                        phl ! Input ? f: eframe[IsSABMorSABME(f)];
                        exit(collis(f, fr), f) >>
                    accept coll: Bool, f: eframe in
                        [not(coll)] ->
                        (
                                pp ! Succ(0) of Nat;
                                p ! Succ(Succ(0)) of Nat;
                                phl ! Out ! m(make_dm(flagging, from,
                                    ctl_uframe(pf(fr)), fc, flagging), noinfor,
                                    correct, notless, noabort);
                                exit(true of Bool, false, f)
                        )
                    []
                        [coll] ->
                            (
                                collision[dl, phl, pp, p, t](fr, from, to, n2,
                                    k, tmode, resolvecollis) >>
                                accept b1: Bool, b2: Bool in
                                exit(b1, b2, f)
                        )
                    )
                )
        []
            (
                    phl ! Input ? f: eframe[not(IsSABMorSABME(f))];
                    (
                        issabm[dl, phl, pp, p, t](fr, from, to, n2, k, tmode,
                            resolvecollis)
                []
```

```
                    pp ? n: Nat;
                        exit(any Bool, any Bool, any eframe)
                )
                )
            )
        [>
            (
                p ! Succ(Succ(0)) of Nat;
                phl ! Input ? f: eframe;
                (
                    exit(any Bool,any Bool,any eframe)
            []
                wwaitua[phl]
            )
            )
    )
        [>
        (
            p ! Succ(0) of Nat;
            phl ! Input ? f: eframe;
            issabm[dl, phl, pp, p, t](fr, from, to, n2, k, tmode,
                resolvecollis)
    )
    )
endproc (* issabm *)

process isdisc[dl,phl,pp,p,t](fr: eframe, from: adress, to: adress,
        n2: Nat, k: Nat,tmode: mmode, resolvecollis: Nat) :
        exit(Bool, Bool, eframe) :=
(
        (
            (
                (
                    phl ! Input ? f: eframe[not(IsDISC(f))];
                    (
                        isdisc[dl, phl, pp, p, t](fr, from, to, n2, k, tmode,
                            resolvecollis)
                []
                    pp ? n: Nat;
                        exit(any Bool, any Bool, any eframe)
                )
                )
            )
        []
        (
                (
                    phl ! Input ? f: eframe[IsDISC(f)];
                    exit(collis(f, fr), f) >>
                accept coll: Bool,f: eframe in
                (
                        [not(coll)] ->
                    (
                            pp ! Succ(0) of Nat;
                            p ! Succ(Succ(0)) of Nat;
                        phl ! Out ! m(make_dm(flagging, from,
                            ctl_uframe(pf(fr)), fc, flagging), noinfor,
                            correct, notless, noabort);
                        exit(true of Bool, false, f)
                    )
                []
                    [coll] ->
                        (
                        collision[dl, phl, pp, p, t](fr, from, to, n2, k,
                            tmode, resolvecollis) >>
                        accept b1: Bool, b2: Bool in
                        exit(b1, b2, f)
                        )
                )
                )
            )
        )
    [>
        (
            p ! Succ(Succ(0)) of Nat;
            phl ! Input ? f: eframe;
            (
                exit(any Bool,any Bool,any eframe)
        []
            wwaitua[phl]
        )
        )
    )
    [>
    (
        p ! Succ(0) of Nat;
        phl ! Input ? f: eframe;
        isdisc[dl, phl, pp, p, t](fr, from, to, n2, k, tmode, resolvecollis)
    )
endproc (* isdisc *)

    process collision[dl, phl, pp, p, t](f: eframe, from: adress,
        to: adress,n2: Nat,k: Nat,tmode: mmode,resolvecollis: Nat) :
        exit(Bool,Bool) :=

    pp ! Succ(0) of Nat;
        p ! Succ(Succ(0)) of Nat;
        phl ! Out ! m(make_ua(flagging, from, ctl_uframe(pf(f)), fc, flagging),
            noinfor, correct, notless, noabort);
        (
            [resolvecollis eq 0 of Nat] ->
            waitua[dl, phl, pp, p, t](f, from, to, n2, k, tmode,
                    resolvecollis)
    []
        [resolvecollis eq Succ(0) of Nat] ->
            waituaortimeout[dl, phl, pp, p, t]
    []
        [resolvecollis eq Succ(Succ(0)) of Nat] ->
            donotwait
    )
    endproc (* collision *)

process waitua[dl, phl, pp, p, t](f: eframe,from: adress, to: adress,
        n2: Nat, k: Nat, tmode: mmode, resolvecollis: Nat) :
        exit(Bool, Bool) :=

        (
            t ! start;
            waitua1[dl, phl, pp, p, t]
        )
    [>
        (
            t ! expired;
            calloption1[dl, phl, t](f, from, to, n2, k, tmode,
                resolvecollis)
    )
    endproc (* waitua *)

process waitua1[dl, phl, pp, p, t] : exit(Bool, Bool) :=
        phl ! Input ? sframe: eframe [IsUA_F1(sframe)];
        t ! reset;
        exit(true of Bool, false)
[]
    (* Ignore received frame if not UA with F=1 *)
        phl ! Input ? sframe: eframe [not(IsUA_F1(sframe))];
        waitua1[dl,phl,pp,p,t]
endproc  (* waitua1 *)

    process waituaortimeout[dl, phl, pp, p, t] : exit(Bool, Bool) :=
        (
            t ! start;
            waitua1[dl, phl, pp, p, t]
    )
    [>
    (
            t ! expired;    (* Returns true of Bool if timeout occurs *)
        exit(true of Bool, false)
        )
endproc  (* waituaortimeout  *)

    process donotwait : exit(Bool, Bool) :=
    exit(true of Bool, false)
    endproc (* donotwait *)

process sendcde[phl, pp, p, t] (f: eframe, init: Nat, n2: Nat) :
        exit(Bool, Bool, eframe) :=
    (
            [init lt n2] ->
        (
                (
                    p ! Succ(0) of Nat;
                    phl ! Out ! f;
                t ! start;
                synchresp[phl, pp, t, p]
            )
                [>
                (
                    t ! expired; exit >>
                    sendcde[phl, pp, p, t](f, Succ(init), n2)
            )
            )
   []
        [init eq n2] ->
            (
                p ! 0 of Bit;
                exit(false, false, m(make_ua(flagging, a, ctl_uframe(1), fc,
                    flagging), noinfor, correct, notless, noabort))
        )
       )

   where
       process synchresp[phl, pp, t, p] : exit(Bool, Bool, eframe) :=
       (
           (
                   pp ! 0 of Nat;
                   t ! reset;
               exit(any Bool, any Bool, any eframe)
           )
           []
           (
                   pp ! Succ(0) of Nat;
               t ! reset;
               p ! Succ(Succ(0)) of Nat;
               phl ! Input ? fram: eframe;
               (
                       exit(any Bool, any Bool, any eframe)
               []
                   wwaitua[phl]
               )
           )
            )
        []
            (
                phl ! Input ? fram: eframe;
            synchresp[phl,pp,t,p]
        )
        endproc (* synchresp *)
    endproc (* sendcde *)

process calloption1[dl, phl, t] (fr: eframe, from, to: adress, n2,
        k: Nat, tmode: mmode, resolvecollis: Nat) : exit(Bool, Bool) :=

    hide pp, p in
        sendcde[phl, pp, p, t](fr, 0 of Nat, n2)
        |[phl, pp, p]|
        waitresp[dl,phl,pp,p, t](fr, from, to, n2, k, tmode, resolvecollis) >>
        accept b1: Bool, b2: Bool, f: eframe in
        (
            [is_dm(get_f(f)) and is_sabm(get_f(fr))] ->
            (
                i; calloption1[dl, phl, t](fr, from, to, n2, k, tmode,
                    resolvecollis)
            []
                i; exit(b1, b2)
            )
        []
            [not(is_dm(get_f(f)) and is_sabm(get_f(fr)))] ->
                exit(b1, b2)
        )

    where
        process waitresp[dl, phl, pp, p, t](f : eframe, from, to : adress,
            n2, k : Nat, tmode : mmode, resolvecollis : Nat) :
            exit(Bool, Bool, eframe) :=
    (
            p ! Succ(0) of Nat;
            phl ! Out ? f : eframe;
            waitingresponse[dl, phl, pp, p, t](f, from, to, n2, k, tmode,
                resolvecollis)
        )
        [>
            p ! 0 of Bit;
            exit(any Bool, any Bool, any eframe)

        where
            process waitingresponse[dl, phl, pp, p, t](f: eframe, from,
                to: adress, n2: Nat, k: Nat, tmode: mmode,
                resolvecollis: Nat): exit(Bool, Bool, eframe) :=
            (
                issabm[dl, phl, pp, p, t](f, from, to, n2, k, tmode,
                    resolvecollis)
                |[phl, pp, p]|
                isdisc[dl, phl, pp, p, t](f, from, to, n2, k, tmode,
                    resolvecollis)
                |[phl, pp, p]|
                notsabmdiscuaf1dmf1[phl,pp,p]
                |[phl, pp, p]|
                isuaf1[phl, pp, p]
                |[phl, pp, p]|
                isdmf1[phl, pp ,p](f)
            )
            endproc (* waitingresponse *)
        endproc (* waitresp *)
    endproc (* calloption1 *)

    process dataphase[dl,phl,pp,t](notwait: Bool, from: adress,
        to: adress, n2: Nat, k: Nat, tmode: mmode, q: pqueue,
        qdls: pqueue, qdlr: pacqueue, resolvecollis: Nat) : noexit :=
    hide p in
(
    (
            inforphase[dl, phl, p, t](notwait, false, 0 of Nat, 0 of Nat, from,
                to, n2, k, tmode, 0 of Nat, 0 of Nat, 0 of Nat, q, qdls, false,
                false, false, false, qdlr, false)
    [>
            linkreset[dl,phl,p,pp,t](from,to,n2,k,tmode,resolvecollis)
    )
    |[p]|
        causelinkreset[p]
    ) >> stop

    where
    process receiveinvalid1[phl] : exit :=
        phl ! Input ? f: eframe[not(is_valid(f))];
        exit
    endproc (* receiveinvalid *)

    process receiveinvalid[phl](vs, vr, lu, vsp: Nat, q, qdls: pqueue,
            qdlr: pacqueue, starttime, rbusy, lbusy, chkpt, retrans, reject,
            notwait: Bool, init, k: Nat, to, from: adress, tmode: mmode) :
            exit(Nat, Nat, Nat, Nat, pqueue, pqueue, pacqueue, Bool, Bool,
            Bool, Bool, Bool, Bool, Nat, Bool, Nat, frmrreason, Nat, Bit) :=

        phl ! Input ? f: eframe[not(is_valid(f))];
            exit(vs, vr, lu, vsp, q, qdls, qdlr, starttime, rbusy, lbusy,
                chkpt, retrans, reject, init, notwait, 0 of Nat, nors,
                0 of Nat, 0 of Bit)
        endproc (* receiveinvalid *)

    process causelinkreset[p] : exit :=
        p ! lreset ? n: Nat ? reasonfrmr: frmrreason ? q: pqueue
                ? qdls: pqueue ? qdlr: pacqueue ? b: Bit ? vs: Nat
                ? vr: Nat ? lu: Nat;
            p ! causelreset ! n  ! reasonfrmr ! q ! qdls ! qdlr ! b  ! vs
                ! vr ! lu;
            exit
    endproc (* causelinkreset *)

    process linkreset[dl, phl, p, pp, t](from, to: adress, n2,
        k: Nat, tmode: mmode, resolvecollis: Nat) : noexit :=

        p ! causelreset ? n: Nat ? reasonfrmr: frmrreason ? q: pqueue
                ? qdls: pqueue ? qdlr: pacqueue ? b: Bit ? vs: Nat
                ? vr: Nat ? lu: Nat;
            (
                [n eq 0 of Nat] ->
            (
                    calloption1[dl,phl,t](m(make_sabm(flagging, to,
                        ctl_uframe(1), fc, flagging), noinfor, correct,
                        notless, noabort), from, to, n2, k, tmode,
                        resolvecollis) >>
                accept ok: Bool,notwait: Bool in
                (
                        [ok] ->
                            dataphase[dl, phl, pp, t] (notwait, from, to, n2,
                        k, tmode, q, qdls, qdlr, resolvecollis)
            []
                [not(ok)] ->
                    pp ! termine;
                    stop
            )
        )
        []
        [(n eq Succ(0))] ->    (* frmr sent *)
        (
                i; sendfrmr1[dl, phl, pp, p, t](vs, vr, lu, 0 of Nat, from,
                    to, n2, k, tmode, reasonfrmr, q, qdls, qdlr,
                    resolvecollis)
        []
                i; sendfrmr2[dl, phl, pp, p](vs, vr, lu, from, to, n2, k,
                    tmode, reasonfrmr, q, qdls, qdlr, resolvecollis)
        )
    []
        [((n eq Succ(Succ(0)) of Nat) or
                (n eq Succ(Succ(Succ(0))) of Nat))] ->
                rcvfrmroruaordm[dl, phl, pp, t](from, to, n2, k, tmode, q,
                    qdls, qdlr, resolvecollis)
        []
        [n eq (Succ(Succ(0)) * Succ(Succ(0)) of Nat)] ->
            rcvsabm[dl, phl, pp, t](from, to, n2, k, tmode, q, qdls,
                    qdlr, b, resolvecollis)
    )

    where
    process rcvfrmroruaordm[dl, phl, pp, t](from, to: adress,
            n2, k: Nat, tmode: mmode, q, qdls: pqueue, qdlr: pacqueue,
            resolvecollis: Nat) : noexit :=
    (* ask peer entity to initiate link set up and enter disconnected
           phase *)
        (
            i; phl ! Out ! m(make_dm(flagging, from, ctl_uframe(0), fc,
                flagging), noinfor, correct, notless, noabort);
            pp ! termine;
            stop
    )
    []
        (* initiate link reset procedure *)
        i; calloption1[dl, phl, t](m(make_sabm(flagging, to,
                ctl_uframe(1), fc, flagging), noinfor, correct, notless,
                noabort), from, to, n2, k, tmode, resolvecollis) >>
        accept ok: Bool, notwait: Bool in
        (
            [ok] ->
                    dataphase[dl, phl, pp, t](notwait, from, to, n2, k,
                        tmode, q, qdls, qdlr, resolvecollis)
        []
            [not(ok)] ->
                pp ! termine;
                stop
        )
    endproc (* rcvfrmroruaordm *)

        process rcvsabm[dl, phl, pp, t](from, to: adress,
            n2, k: Nat, tmode: mmode, q: pqueue, qdls: pqueue,
            qdlr: pacqueue, b: Bit, resolvecollis: Nat) : noexit :=
    (* remain in information transfer phase *)
    (
            i; phl ! Out ! m(make_ua(flagging, from, ctl_uframe(b), fc,
                flagging), noinfor, correct, notless, noabort);
            dataphase[dl, phl, pp, t](false, from, to, n2, k, tmode, q,
                qdls, qdlr, resolvecollis)
    )
    []
    (* enters disconnected *)
        i; phl ! Out ! m(make_dm(flagging, to, ctl_uframe(b), fc,
                flagging), noinfor, correct, notless, noabort);
        pp ! termine;
        stop
        endproc   (* rcvsabm *)

        process sendfrmr1[dl, phl, pp, p, t](vs, vr, lu, init: Nat,
            from, to: adress, n2, k: Nat, tmode: mmode,
            reasonfrmr: frmrreason, q, qdls: pqueue,
            qdlr: pacqueue, resolvecollis: Nat) : noexit :=

        [init lt n2] ->
        (
                (
                    phl ! Out ! m(make_frmr(flagging, to, ctl_uframe(1),
                        reasonfrmr, fc, flagging), noinfor, correct,
                        notless, noabort);
                    t ! start;
                waitrespfrmr[dl, phl, pp, t, p](vs, vr, lu,
                        0 of Nat, from, to, n2, k, tmode, reasonfrmr,
                        q, qdls, qdlr, resolvecollis)
            )
            [>
                    t ! expired; exit >>
                sendfrmr1[dl, phl, pp, p, t](vs, vr, lu,
                        Succ(init), from, to, n2, k, tmode, reasonfrmr,
                        q, qdls, qdlr, resolvecollis)
        )
        []
        (* if FRMR sent n2 times without response reset the link *)
        [init eq n2] ->
        (
                calloption1[dl, phl, t](m(make_sabm(flagging, to,
                    ctl_uframe(1), fc, flagging), noinfor, correct,
                    notless, noabort), from, to, n2, k, tmode,
                    resolvecollis) >>
            accept ok: Bool,notwait: Bool in
            (
                [ok] ->
```

```
                         dataphase[dl, phl, pp, t](notwait, from, to, n2, k,
                             tmode, q, qdls, qdlr, resolvecollis)
                 []
                     [not(ok)] -> (* enter disconnected phase *)
                         pp ! termine;
                         stop
                 )
             )
             endproc (* sendfrmr1 *)

     process sendfrmr2[dl,phl,pp,p] (vs: Nat,vr: Nat,lu: Nat,from: adress,to: adress,n2: Nat,
        k: Nat,tmode: mmode,reasonfrmr: frmrreason,q: pqueue,qdls: pqueue,qdlr: pacqueue,
        resolvecollis: Nat) : noexit :=
      hide t in
       (phl ! Out ! m(make_frmr(flagging,to,ctl_uframe(1),reasonfrmr,fc,flagging),noinfor,correct,
                             notless,noabort)
       ;waitrespfrmr[dl,phl,pp,t,p](vs,vr,lu,Succ(0) of Nat,from,to,n2,k,tmode,reasonfrmr,q,qdls,
                             qdlr,resolvecollis)
      )
     endproc (* sendfrmr2 *)

     process waitrespfrmr[dl,phl,pp,t,p](vs: Nat,vr: Nat,lu: Nat,call: Nat,from: adress,
                to: adress,n2: Nat,k: Nat,tmode: mmode,reasonfrmr: frmrreason,q: pqueue,
                qdls: pqueue,qdlr: pacqueue,resolvecollis: Nat)
          : noexit :=
       (receiveinvalid1[phl]
        >> waitrespfrmr[dl,phl,pp,t,p](vs,vr,lu,call,from,to,n2,k,tmode,reasonfrmr,q,qdls,qdlr,
                                     resolvecollis)
      )
      []
      (sendfrmr[phl](vs,vr,lu,k,from,to,n2,q,qdls,qdlr)
       >> accept ind: Nat,rs: frmrreason,q: pqueue,qdls: pqueue,qdlr: pacqueue,b: Bit in
           phl ! Out ! m(make_frmr(flagging,to,ctl_uframe(1),reasonfrmr,fc,flagging),noinfor,
                                     correct,notless,noabort)
        ;waitrespfrmr[dl,phl,pp,t,p](vs,vr,lu,call,from,to,n2,k,tmode,reasonfrmr,q,qdls,qdlr,
                                     resolvecollis)
      )
      []
      ((phl ! Input ?  f: eframe[(correct(f) and (eq(adrs(f),from))) and
                          (pf(f) eq 1 of Bit)];exit
       |[phl]|
       notfrmrsabmdmdisc[phl]
      ) >>
         phl ! Out ! m(make_frmr(flagging,from,ctl_uframe(1),reasonfrmr,fc,flagging),noinfor,
                                     correct,notless,noabort)
        ;waitrespfrmr[dl,phl,pp,t,p](vs,vr,lu,call,from,to,n2,k,tmode,reasonfrmr,q,qdls,qdlr,
                                     resolvecollis)
      )
      []
      ((phl ! Input ? f: eframe[correct(f) and ((eq(adrs(f),from)) or ((eq(adrs(f),to)) and
                         (pf(f) eq 0 of Bit)))];exit
        |[phl]|
        notfrmrsabmdmdisc[phl]
       ) >> waitrespfrmr[dl,phl,pp,t,p](vs,vr,lu,call,from,to,n2,k,tmode,reasonfrmr,q,qdls,qdlr,
                                        resolvecollis)
      )
      []
     ((phl ! Input ? f: eframe[correct(f)];exit(f)
       |[phl]|
       frmrsabmdmdisc[phl]
      ) >> accept f: eframe in
         (([call eq 0 of Nat] ->
                                ( t ! reset
                                  ;exit
                                )
          []
          [call eq Succ(0) of Nat] ->
                         exit
         )
         >> ([(is_sabm(get_f(f)) or is_sabme(get_f(f)))] ->
(* ---------------------------------------------------------------------------
             case (select FR in f) in
                  make_sabm (...)
                | make_sabme (...) ->
========================================================================== *)
           ( (i;phl ! Out ! m(make_ua(flagging,to,ctl_uframe(pf(f)),fc,flagging),noinfor,
                                        correct,notless,noabort)
             ;dataphase[dl,phl,pp,t](false,from,to,n2,k,tmode,q,qdls,qdlr,resolvecollis)
             )
            []
            (i;phl ! Out ! m(make_dm(flagging,to,ctl_uframe(pf(f)),fc,flagging),noinfor,
                                        correct,notless,noabort)
             ;pp ! termine
             ;stop
             )

          )
          []
          ([is_dm(get_f(f)) and (pf(f) eq 1 of Bit)] ->
(* ---------------------------------------------------------------------------
                | make_dm (...) when (pf(f) eq 1 of Bit) ->
========================================================================== *)
                (waitrespfrmr[dl,phl,pp,t,p](vs,vr,lu,call,from,to,n2,k,tmode,reasonfrmr,q,qdls,
                                        qdlr,resolvecollis))
          )
          []
          ([is_dm(get_f(f)) and (pf(f) eq 0 of Bit)] ->
                (* remote entitty asks local entity to initiate lin set up *)
(* ---------------------------------------------------------------------------
                | make_dm (...) when (pf(f) eq 0 of Bit) ->
========================================================================== *)
            ((i;phl ! Out ! m(make_disc(flagging,to,ctl_uframe(1),fc,flagging),noinfor,
                                        correct,notless,noabort)
             ;pp ! termine
             ;stop
             )
             []
            (i;calloption1[dl,phl,t](m(make_sabm(flagging,to,ctl_uframe(1),fc,flagging),noinfor,
                              correct,notless,noabort), from,to,n2,k,tmode,resolvecollis)
             >> accept ok: Bool,notwait: Bool in
               ([not(ok)] ->
                          ( pp ! termine
                           ;stop
                          )
                []
                [ok] ->
                  ((hide pp in
                    (dataphase[dl,phl,pp,t](notwait,from,to,n2,k,tmode,q,qdls,qdlr,resolvecollis)
                     [> terminationphase[dl,phl,pp,t](from,to,n2,k,tmode,resolvecollis)
                    )
                    |[pp]|
                    goterm[pp]
                  ) >> stop
                  )
               )
          )
         ))
        []
        ([is_frmr(get_f(f))] ->          (* collision *)
(* ---------------------------------------------------------------------------
                | make_frmr (...) ->
========================================================================== *)
           (* send DM to ask remote entitty link set up and enter disconnected phase *)
           (i;phl ! Out ! m(make_dm(flagging,from,ctl_uframe(0),fc,flagging),noinfor,
                              correct,notless,noabort)
           ;pp ! termine
           ;stop
           )
          []
          (* or cause link reset *)
          i;calloption1[dl,phl,t](m(make_sabm(flagging,to,ctl_uframe(1),fc,flagging),noinfor,
                              correct,notless,noabort),from,to,n2,k,tmode,resolvecollis)
          >> accept ok: Bool,notwait: Bool in
               ([ok] ->
                     dataphase[dl,phl,pp,t](notwait,from,to,n2,k,tmode,q,qdls,qdlr,resolvecollis)
                []
                [not(ok)] ->  (* enter disconnect state *)
                    pp ! termine
                   ;stop
               )
          )))
(* ---------------------------------------------------------------------------
                endcase
========================================================================== *)
        )
      where
         process frmrsabmdmdisc[phl] : exit(eframe) :=
               phl ! Input ? f: eframe[((is_sabm(get_f(f)) or is_sabme(get_f(f))) or
                                       (is_frmr(get_f(f)) or is_dm(get_f(f))))]
              ;exit(f)
            endproc (* frmrsabmdm  *)
          process notfrmrsabmdmdisc[phl] : exit :=
            phl ! Input ? f: eframe[not(((is_sabm(get_f(f)) or is_sabme(get_f(f))) or
                                 (is_frmr(get_f(f)) or is_dm(get_f(f)))) or is_disc(get_f(f)))]
           ;exit
          endproc (* notfrmrsabmdm  *)
   endproc  (* waitrespfrmr *)

    endproc (* linkreset *)

        process sendfrmr[phl](vs, vr, lu, k: Nat, from, to: adress,
            n2: Nat, q, qdls: pqueue, qdlr: pacqueue) : exit(Nat,
            frmrreason, pqueue, pqueue, pacqueue, Bit) :=
            (
                (
                    (
                        phl ! Input ? f: eframe[(is_valid(f) and not_abort(f)) and
                            (((not(is_i(get_f(f)))) and (is_inforr(f))) or
                            (not(corrl(f))))];
                        exit(f)
                )
                []
                (
                        phl ! Input ? f: eframe[(is_valid(f) and not_abort(f)) and
                            is_undef_ctl(get_ctl(get_f(f)))];
                    exit(f)
                )
                []
                    (
                        phl ! Input ? f: eframe[(is_valid(f) and not_abort(f)) and
                            notvalid_nrf(f, vs, lu, k)];
                        exit(f)            (* nr of frame not valid *)
                )
                []
                    (
                        phl ! Input ? f: eframe[(is_valid(f) and not_abort(f)) and
                            (is_i(get_f(f)) and (not(corrl(f))))];
                        exit(f)
                    )
                )
            >> accept f: eframe in
                (
                    (
                        (
                            [(is_valid(f) and not_abort(f)) and
                                (((not(is_i(get_f(f)))) and (is_inforr(f))) or
                                (not(corrl(f))))] ->
                            (
                                    exit(1 of Bit, any Bit, any Bit, 1 of Bit,
                                        any Bit) (* returns x=1, w=1 *)
                            )
                    []
                        [not((is_valid(f) and not_abort(f)) and
                                (((not(is_i(get_f(f)))) and (is_inforr(f))) or
                                (not(corrl(f)))))] ->
            (
                [(is_valid(f) and not_abort(f)) and
                    is_undef_ctl(get_ctl(get_f(f)))] ->
                (
                    exit(0 of Bit, any Bit, any Bit, 1 of Bit,
                        any Bit) (* returns x=0, w=1 *)
                )
            []
                [(is_valid(f) and not_abort(f)) and
                    is_undef_ctl(get_ctl(get_f(f)))] ->
                (
                    exit(0 of Bit, any Bit, any Bit, 0 of Bit,
                        any Bit) (* returns x=0, w=0 *)
                )
            )
)
    |||
(
        [(is_valid(f) and not_abort(f)) and notvalid_nrf(f,
            vs, lu, k)] ->
        (
            exit(any Bit, any Bit, 1 of Bit, any Bit,
                any Bit) (* returns z=1 *)
        )
[]
    [not((is_valid(f) and not_abort(f)) and
            notvalid_nrf(f, vs, lu, k))] ->
    (
            exit(any Bit, any Bit, 0 of Bit, any Bit,
                any Bit) (* returns z =0 *)
        )
)
    |||
    (
        [(is_valid(f) and not_abort(f)) and (is_i(get_f(f))
            and (not(corrl(f))))] ->
        (
            exit(any Bit, 1 of Bit, any Bit, any Bit,
            any Bit) (* returns y=1 *)
        )
    []
        [not((is_valid(f) and not_abort(f)) and
            (is_i(get_f(f)) and (not(corrl(f)))))] ->
        (
            exit(any Bit, 0 of Bit, any Bit, any Bit,
                any Bit) (* returns y = 0 *)
        )
    )
    |||
    (
    (* command frame so c/r = 0     *)
        [(pf(f) eq 1) and (eq(adrs(f), from))] ->
        (
            exit(any Bit, any Bit, any Bit, any Bit,
                                0 of Bit)
                        )
                    []
                    (* response frame so c/r = 1 *)
                        [not((pf(f) eq 1) and (eq(adrs(f), from)))] ->
                        (
                            exit(any Bit, any Bit, any Bit, any Bit,
                                1 of Bit)
                        )
                    )
                ) >>
                accept x, y, z, w, cr : Bit in
                exit(Succ(0) of Nat, make_reason(ctrl(f), 0 of Bit, vs, cr,
                    vr, x, y, z, w, 0 of Bit), q, qdls, qdlr, 1 of Bit)
            )
        )
    endproc  (* sendfrmr *)

    process inforphase[dl, phl, p, t] (notwait: Bool, reject: Bool,
        vsp, init: Nat, from, to: adress, n2, k: Nat, tmode: mmode,
        vs, vr, lu: Nat, q, qdls: pqueue, starttime, rbusy, lbusy,
        chkpt: Bool, qdlr: pacqueue, retrans: Bool) : exit :=

        infphase[dl, phl, p, t](notwait, false, 0 of Nat, 0 of Nat, from,
            to, n2, k, tmode, 0 of Nat, 0 of Nat, 0 of Nat, q, qdls, false,
            false, false, false, qdlr, false)

    where
        process infphase[dl, phl, p, t](notwait: Bool, reject: Bool,
            vsp, init: Nat, from, to: adress, n2: Nat,
            k: Nat, tmode: mmode, vs, vr, lu: Nat, q, qdls: pqueue,
            starttime, rbusy, lbusy, chkpt: Bool, qdlr: pacqueue,
            retrans: Bool) : exit :=

        (
            comx25[dl] (vs, vr, lu, vsp, q, qdls, qdlr, starttime, rbusy,
                lbusy, chkpt, retrans, reject, notwait, init, k, to, from,
                tmode)
        []
            sendframe[phl, t](vs, vr, lu, vsp, q, qdls, qdlr, starttime,
                rbusy, lbusy, chkpt, retrans, reject, notwait, init, k, to,
                from, tmode)
        []
            receiveframe[phl, t](vs, vr, lu, vsp, q, qdls, qdlr, starttime,
                rbusy, lbusy, chkpt, retrans, reject, notwait, init, k, to,
                from, tmode)
        []
             timeout[phl,t] (n2, vs, vr, lu, vsp, q, qdls, qdlr, starttime,
                 rbusy, lbusy, chkpt, retrans, reject, notwait, init, k,
                 to, from, tmode)
        )
        [>
            rcvtoreset[phl,p](notwait, chkpt, from, to, n2, k, tmode, vs,
                vr, lu, q, qdls, qdlr, rbusy) >>
                accept vs, vr, lu, vsp: Nat, q, qdls: pqueue, qdlr:
                pacqueue, starttime, rbusy, lbusy, chkpt, retrans, reject:
                Bool, init: Nat, notwait: Bool, typ: Nat, rs: frmrreason,
                ind: Nat, b: Bit in
            (
                [typ eq 0 of Nat] ->
========================================================================== *)
                (
                    infphase[dl, phl, p, t](notwait, reject, vsp, init, from,
                        to, n2, k, tmode, vs, vr, lu, q, qdls, starttime,
                        rbusy, lbusy, chkpt, qdlr, retrans)
                )
            []
                [typ eq Succ(0) of Nat] ->
                (
                    hide t in
                        t ! start;
                        infphase[dl, phl, p, t](notwait, reject, vsp, init,
                            from, to, n2, k, tmode, vs, vr, lu, q, qdls,
                            starttime, rbusy, lbusy, chkpt, qdlr, retrans)
                )
            []
                [typ eq Succ(Succ(0)) of Nat] ->
                    t ! start;
                    infphase[dl, phl, p, t](notwait, reject, vsp, init, from,
                        to, n2, k, tmode, vs, vr, lu, q, qdls, starttime,
                        rbusy, lbusy, chkpt, qdlr, retrans)
            []
                [typ eq Succ(Succ(Succ(0))) of Nat] ->
                    infphase[dl, phl, p, t](notwait, reject, vsp, init, from,
                        to, n2, k, tmode, vs, vr, lu, q, qdls, starttime,
                        rbusy, lbusy, chkpt, qdlr, retrans)
            []
                [typ eq Succ(Succ(Succ(Succ(0)))) of Nat] ->
                (
                    p ! lreset ! ind ! rs ! empty of pqueue ! qdls ! qdlr !
                        b !  vs ! vr ! lu;
                    stop
                )
             )
(* ------------------------------------------------------------------------
        (case typ in
          0 -> ( infphase[dl, phl, p, t](notwait, reject, vsp, init, from,
                        to, n2, k, tmode, vs, vr, lu, q, qdls, starttime,
                        rbusy, lbusy, chkpt, qdlr, retrans)
                )
        | 1 -> ( hide t in
                        t ! start;
                        infphase[dl, phl, p, t](notwait, reject, vsp, init,
                            from, to, n2, k, tmode, vs, vr, lu, q, qdls,
                            starttime, rbusy, lbusy, chkpt, qdlr, retrans)
                )
        | 2 -> t ! start;
                    infphase[dl, phl, p, t](notwait, reject, vsp, init, from,
                  to, n2, k, tmode, vs, vr, lu, q, qdls, starttime,
                  rbusy, lbusy, chkpt, qdlr, retrans)
        | 3 -> infphase[dl, phl, p, t](notwait, reject, vsp, init, from,
                  to, n2, k, tmode, vs, vr, lu, q, qdls, starttime,
                  rbusy, lbusy, chkpt, qdlr, retrans)
        | 4 -> ( p ! lreset ! ind ! rs ! empty of pqueue ! qdls ! qdlr !
                  b !  vs ! vr ! lu;
              stop
          )
        endcase
        )
========================================================================== *)

             where
                process comx25[dl](vs, vr, lu, vsp: Nat, q, qdls: pqueue,
                    qdlr: pacqueue, starttime, rbusy, lbusy, chkpt, retrans,
                    reject, notwait: Bool, init, k: Nat, to, from: adress,
                    tmode: mmode) : exit(Nat, Nat, Nat, Nat, pqueue, pqueue,
                    pacqueue, Bool, Bool, Bool, Bool, Bool, Bool, Nat, Bool,
                    Nat, frmrreason, Nat, Bit) :=

                    recvpack[dl](vs, vr, lu, vsp, q, qdls, qdlr, starttime,
                        rbusy, lbusy, chkpt, retrans, reject, notwait, init,
                        k, to, from, tmode)
                []
                    deliver[dl] (vs, vr, lu, vsp, q, qdls, qdlr, starttime,
                        rbusy, lbusy, chkpt, retrans, reject, notwait, init,
                        k, to, from, tmode)
                endproc (* comx25 *)

                process sendframe[phl,t](vs, vr, lu, vsp: Nat, q, qdls:
                    pqueue, qdlr: pacqueue, starttime, rbusy, lbusy, chkpt,
                    retrans, reject, notwait: Bool, init, k: Nat, to, from:
                    adress, tmode: mmode) : exit(Nat, Nat, Nat, Nat, pqueue,
                    pqueue, pacqueue, Bool, Bool, Bool, Bool, Bool, Bool, Nat,
                    Bool, Nat, frmrreason, Nat, Bit) :=

                    sendi[phl, t](vs, vr, lu, vsp, q, qdls, qdlr, starttime,
                        rbusy, lbusy, chkpt, retrans, reject, notwait, init,
                        k, to, from, tmode)
                []
                    sendrr[phl](vs, vr, lu, vsp, q, qdls, qdlr, starttime,
                        rbusy, lbusy, chkpt, retrans, reject, notwait, init,
                        k, to, from, tmode)
                []
                    sendrnr[phl](vs, vr, lu, vsp, q, qdls, qdlr, starttime,
                        rbusy, lbusy, chkpt, retrans, reject, notwait, init,
                        k, to, from, tmode)
                endproc (* sendframe *)

                process receiveframe[phl, t](vs, vr, lu, vsp: Nat, q, qdls:
                    pqueue, qdlr: pacqueue, starttime, rbusy, lbusy, chkpt,
                    retrans, reject, notwait: Bool, init, k: Nat, to, from:
                    adress, tmode: mmode) :  exit(Nat, Nat, Nat, Nat, pqueue,
                      pqueue, pacqueue, Bool, Bool, Bool, Bool, Bool, Bool, Nat,
                      Bool, Nat, frmrreason, Nat, Bit)  :=

                      receiveinvalid[phl](vs, vr, lu, vsp, q, qdls, qdlr,
                          starttime, rbusy, lbusy, chkpt, retrans, reject,
                          notwait, init, k, to, from, tmode)
                  []
                      receivevalidi[phl, t] (vs, vr, lu, vsp, q, qdls, qdlr,
                          starttime, rbusy, lbusy, chkpt, retrans, reject,
                          notwait, init, k, to, from, tmode)
                  []
                      rcviandbusy[phl, t](vs, vr, lu, vsp, q, qdls, qdlr,
                          starttime, rbusy, lbusy, chkpt, retrans, reject,
                          notwait, init, k, to, from, tmode)
                  []
                      receivernr[phl, t] (vs, vr, lu, vsp, q, qdls, qdlr,
                          starttime, rbusy, lbusy, chkpt, retrans, reject,
                          notwait, init, k, to, from, tmode)
                  []
                      receiverr[phl, t](vs, vr, lu, vsp, q, qdls, qdlr,
                          starttime, rbusy, lbusy, chkpt, retrans, reject,
                          notwait, init, k, to, from, tmode)
                  []
                      receiverej[phl](vs, vr, lu, vsp, q, qdls, qdlr, starttime,
                          rbusy, lbusy, chkpt, retrans, reject, notwait, init,
                          k, to, from, tmode)
                  []
                      sendrej[phl](vs, vr, lu, vsp, q, qdls, qdlr, starttime,
                          rbusy, lbusy, chkpt, retrans, reject, notwait, init,
                          k, to, from, tmode)
                   endproc  (* receiveframe *)

      process rcvtoreset[phl,p] (notwait: Bool,chkpt: Bool, from: adress,to
: adress,n2: Nat ,k: Nat,tmode :       mmode,
      vs: Nat,vr: Nat,lu: Nat,q: pqueue,qdls: pqueue,qdlr : pacqueue,rbusy: Bool)
      : noexit :=

        (receiveunsolicit[phl](notwait,chkpt,to,q,qdls,qdlr)
         []
         receivesabm[phl] (q,qdls,qdlr)
         []
         receiveuaordm[phl] (q,qdls,qdlr)
         []
         sendfrmr[phl](vs,vr,lu,k,from,to,n2,q,qdls,qdlr)
         []
         receivefrmr[phl](from,to,n2,k,tmode,vs,vr,lu,q,qdls,qdlr,rbusy)
        ) >> accept ind: Nat,rs: frmrreason,q: pqueue,qdls: pqueue,qdlr:
pacqueue,b: Bit in
               (p ! lreset ! ind ! rs ! empty of pqueue ! qdls ! qdlr ! b ! vs ! vr ! lu
               ;stop
              )
     where
       process receiveunsolicit[phl] (notwait: Bool, chkpt: Bool,to: adress,q: pqueue,
                                        qdls: pqueue,qdlr: pacqueue)
         : exit(Nat,frmrreason,pqueue,pqueue,pacqueue,Bit) :=
          [not(chkpt) and not(notwait)] ->
             (phl ! Input ? f: eframe[(((correct(f) and (not(is_sabm(get_f(f))))) and
                                    (not(is_sabme(get_f(f))))) and
               (not(is_disc(get_f(f)))))
               and ((eq(adrs(f),to)) and (pf(f) eq 1 of Bit))]
             ;exit((Succ(Succ(Succ(0))) of Nat),make_reason(ctrl(f),0 of Bit,0 of Nat,
          0 of Bit,0 of Nat,0 of Bit,0 of Bit, 0 of Bit,0 of Bit,0 of Bit),q,qdls,qdlr,1 of Bit)
             )
        endproc  (* receiveunsolicit *)

        process receivesabm[phl](q: pqueue,qdls: pqueue,qdlr: pacqueue)
          : exit(Nat,frmrreason,pqueue,pqueue,pacqueue,Bit) :=
            phl ! Input ? f: eframe[correct(f) and (is_sabm(get_f(f)) or is_sabme(get_f(f)))]
             ;exit((Succ(Succ(0)) * Succ(Succ(0)) of Nat),make_reason(ctrl(f),0 of Bit,0 of Nat,
               0 of Bit,0 of Nat,0 of Bit,0 of Bit, 0 of Bit,0 of Bit,0 of Bit),q,qdls,qdlr,pf(f))
       endproc  (* receivesabm *)

       process receiveuaordm[phl] (q: pqueue,qdls: pqueue,qdlr: pacqueue)
         : exit(Nat,frmrreason,pqueue,pqueue,pacqueue,Bit) :=
          phl ! Input ? f: eframe[correct(f) and (is_ua(get_f(f)) or is_dm(get_f(f)))]
          ;exit((Succ(Succ(Succ(0))) of Nat),make_reason(ctrl(f),0 of Bit,0 of Nat,
          0 of Bit,0 of Nat,0 of Bit,0 of Bit, 0 of Bit,0 of Bit,0 of Bit),q,qdls,qdlr,1 of Bit)
      endproc (* receiveuaordm *)

      process receivefrmr[phl](from: adress,to: adress,n2: Nat ,k: Nat,tmode: mmode,
        vs: Nat,vr: Nat,lu: Nat,q: pqueue,qdls: pqueue,qdlr: pacqueue,rbusy: Bool)
        : exit(Nat,frmrreason,pqueue,pqueue,pacqueue,Bit) :=
        phl ! Input ? f: eframe[correct(f) and is_frmr(get_f(f))]

         ;exit((Succ(Succ(0)) of Nat),make_reason(ctrl(f),0 of Bit,0 of Nat,
          0 of Bit,0 of Nat,0 of Bit,0 of Bit, 0 of Bit,0 of Bit,0 of Bit),q,qdls,qdlr,1 of Bit)
       endproc (* receivefrmr *)
   endproc (* rcvtoreset *)

       process deliver[dl]  (vs,vr,lu,vsp: Nat,q,qdls: pqueue,qdlr: pacqueue,
        starttime,rbusy,lbusy,chkpt,retrans,reject,notwait: Bool,init,k: Nat,
        to,from: adress,tmode: mmode)
            : exit(Nat,Nat,Nat,Nat,pqueue,pqueue,pacqueue,Bool,Bool,Bool,Bool,Bool,Bool,
                Nat,Bool,Nat,frmrreason,Nat,Bit) :=
        [not(isempty(qdlr))] ->
           (dl ! dl_data_ind(firstone(qdlr))
            ;exit(vs,vr,lu,vsp,q,qdls,removefirst(qdlr),starttime,rbusy,lbusy,
chkpt,retrans,reject,init,notwait,0 of Nat,nors,0 of Nat,0 of Bit)
                )
       endproc
       process retransmission[phl] (vs,vr,lu,vsp: Nat,q,intq,qdls: pqueue,qdlr: pacqueue,
        starttime,rbusy,lbusy,chkpt,retrans,reject,notwait: Bool,init,k: Nat,to,from: adress,
        tmode: mmode)
             : exit(Nat,Nat,Nat,Nat,pqueue,pqueue,pacqueue,Bool,Bool,Bool,Bool,Bool,Bool,
                Nat,Bool,Nat,frmrreason,Nat,Bit) :=
     [not(rbusy)] ->
      ( ([not(isempty(intq))] ->
          (phl ! Out ! m(make_iframe(flagging,to,ctl_iframe(vs,0,(vr mod k)),packe(firstone(intq)),
                   fc,flagging),infor,correct,notless,noabort)
      ;retransmission[phl] ((vs + Succ(0) of Nat),vr,lu,vsp,q,removefirst(intq),qdls,qdlr,
                   starttime,rbusy,lbusy,chkpt,retrans,reject,notwait,init,k,to,from,tmode)
       )
      []
      [isempty(intq)] ->
       (exit(vs,vr,lu,vsp,q,qdls,qdlr,true of Bool,rbusy,lbusy,chkpt,retrans,reject,init,
                   notwait,Succ(0) of Nat,nors,0 of Nat,0 of Bit))
    [> (receiverej[phl] (vs,vr,lu,vsp,q,qdls,qdlr,starttime,rbusy,lbusy,chkpt,retrans,reject,
                   notwait,init,k,to,from,tmode)
        >>
       accept vs: Nat,vr: Nat,lu: Nat,vsp: Nat,q: pqueue,qdls: pqueue,qdlr: pacqueue,
           starttime: Bool,rbusy: Bool,lbusy: Bool,chkpt: Bool,retrans: Bool,
           reject: Bool,init: Nat,notwait: Bool,typ: Nat,rs: frmrreason,ind: Nat,b: Bit in
          (phl ! Out ! m(make_iframe(flagging,to,ctl_iframe(snn(firstone(intq)),0,(vr mod k)),
                       packe(firstone(intq)),fc,flagging),infor,correct,notless,abort)
           ;retransmission[phl] (vs,vr,lu,vsp,q,q,qdls,qdlr,starttime,rbusy,lbusy,chkpt,
                       retrans,reject,notwait,init,k,to,from,tmode)
          )
   )))
  []
  [rbusy] ->
     (exit(vs,vr,lu,vsp,q,qdls,qdlr,true of Bool,rbusy,lbusy,chkpt,retrans,reject,init,notwait,
           Succ(0) of Nat,nors,0 of Nat,0 of Bit))
 endproc

    process  recvpack[dl] (vs,vr,lu,vsp: Nat,q,qdls: pqueue,qdlr: pacqueue,starttime,rbusy,
      lbusy,chkpt,retrans,reject,notwait: Bool,init,k: Nat,to,from: adress,tmode: mmode)
      : exit(Nat,Nat,Nat,Nat,pqueue,pqueue,pacqueue,Bool,Bool,Bool,Bool,Bool,Bool,Nat,Bool,
           Nat,frmrreason,Nat,Bit) :=
     [((Succ(vsp) mod k) ne lu)] ->
        (dl ? p: primitive [is_dl_data_req(p)]
        ;([rbusy and not(starttime)] ->
          (exit(vs,vr,lu,((vsp + Succ(0) mod k) ),q,((mp(get_pack(p),vsp)) +-- qdls) of pqueue,
            qdlr,true of Bool,rbusy,lbusy,chkpt,retrans,reject,init,notwait,(Succ(0) of Nat),
            nors,0 of Nat,0 of Bit))
          []
          [not(rbusy) or starttime] ->
          (exit(vs,vr,lu,((vsp + Succ(0) mod k) ),q,((mp(get_pack(p),vsp)) +-- qdls) of pqueue,
            qdlr,starttime,rbusy,lbusy,chkpt,retrans,reject,init,notwait,0 of Nat,nors,
            0 of Nat,0 of Bit))
        ))
    endproc

   process sendi[phl,t] (vs,vr,lu,vsp: Nat,q,qdls: pqueue,qdlr: pacqueue, starttime,rbusy,
     lbusy,chkpt,retrans,reject,notwait: Bool,init,k: Nat,to,from: adress,tmode: mmode)
     : exit(Nat,Nat,Nat,Nat,pqueue,pqueue,pacqueue,Bool,Bool,Bool,Bool,Bool,Bool,Nat,Bool,Nat,
           frmrreason,Nat,Bit) :=
    ((( (  ( i;phl ! Out ! m(make_iframe(flagging,to,ctl_iframe(vs,0,(vr mod k)),
                   packe(firstone(qdls)),fc,flagging),noinfor,correct,notless,noabort)
      [not(isempty(qdls)) and not(rbusy)]
         ;exit(snn(firstone(qdls)))
       >> accept sn : Nat in
           ([not(starttime)] ->
             (t! start
               ;exit(((vs + Succ(0) of Nat) mod k),vr,lu,vsp,(firstone(qdls) +-- q),
                  removefirst(qdls) of pqueue,qdlr,true of Bool,rbusy,lbusy,chkpt,retrans,
                  reject,init,notwait,0 of Nat,nors,0 of Nat,0 of Bit)
               )
            []
            [starttime] ->
               exit(((vs + Succ(0) of Nat) mod k),vr,lu,vsp,(firstone(qdls) +-- q),
                  removefirst(qdls) of pqueue,qdlr,true of Bool,rbusy,lbusy,chkpt,retrans,
                  reject,init,notwait,0 of Nat,nors,0 of Nat,0 of Bit)
            )))
   []
( (i;phl ! Out ! m(make_iframe(flagging,to,ctl_iframe(vs,1,(vr mod k)),packe(firstone(qdls)),
          fc,flagging),infor,correct,notless,noabort) [not(isempty(qdls)) and not (rbusy)]
     ;exit(snn(firstone(qdls)))
      >> accept sn: Nat in
          ([not(starttime)] ->
            (t! start
             ;exit(((vs + Succ(0) of Nat) mod k),vr,lu,vsp,(firstone(qdls) +-- q),
                  removefirst(qdls) of pqueue,qdlr,true of Bool,rbusy,lbusy,chkpt,retrans,
                  reject,init,notwait,0 of Nat,nors,0 of Nat,0 of Bit)
             )
           []
           [starttime] ->
              exit(((vs + Succ(0) of Nat) mod k),vr,lu,vsp,(firstone(qdls) +-- q),
                  removefirst(qdls) of pqueue,qdlr,true of Bool,rbusy,lbusy,chkpt,retrans,
                  reject,init,notwait,0 of Nat,nors,0 of Nat,0 of Bit)
       ))
   ))
      [> (receiverej[phl] (vs,vr,lu,vsp,q,qdls,qdlr,starttime,rbusy,lbusy,chkpt,retrans,
          reject,notwait,init,k,to,from,tmode)
        >> accept vs: Nat,vr: Nat,lu: Nat,vsp: Nat,q: pqueue,qdls: pqueue,qdlr: pacqueue,
                  starttime: Bool,rbusy: Bool,lbusy: Bool,chkpt: Bool,retrans: Bool,
                  reject: Bool,init: Nat,notwait: Bool,typ: Nat,rs: frmrreason,ind: Nat,
                  b: Bit in
      (phl ! Out ! m(make_iframe(flagging,to,ctl_iframe(vs,1,(vr mod k)),
                          packe(firstone(qdls)),fc,flagging),infor,correct,notless,abort)
       ;retransmission[phl](vs,vr,lu,vsp,q,q,qdls,qdlr,starttime,rbusy,lbusy,chkpt,retrans,
                          reject,notwait,init,k,to,from,tmode)
         )
         )
   )
    [> (receivernr[phl,t] (vs,vr,lu,vsp,q,qdls,qdlr,starttime,rbusy,lbusy,chkpt,retrans,
                          reject,notwait,init,k,to,from,tmode))
        )
   endproc (* sendei *)
 process releasequeue[phl](k: Nat,nr: Nat,q: pqueue) : exit(Nat,pqueue) :=
     [isempty(q)]  -> exit(nr,q)
     []
     [not(isempty(q))] ->
           ([(nr ne (snn(firstone(q))))] ->
                       i
                      ;releasequeue[phl](k,nr,removefirst(q))
           )
```

```
                 []
                 [nr eq (snn(firstone(q)))] ->
                                 exit(nr,q)
       endproc (* releasequeue *)
        process validnrf[phl](vs: Nat,lu: Nat,k: Nat,q: pqueue): exit(Nat,pacqueue,eframe,pqueue) :=
          phl ! Input ? f: eframe [(((nr(f) ge lu) or (nr(f) le (vs mod k))) and
                                 ((vs mod k) lt lu)) or (((nr(f) ge lu) and
                                 (nr(f) le (vs mod k))) and ((vs mod k) ge lu))]
         ;exit(nr(f),any pacqueue,any eframe,q)
       endproc

  process  receivevalidi[phl,t] (vs,vr,lu,vsp: Nat,q,qdls: pqueue,qdlr: pacqueue,starttime,rbusy,
   lbusy,chkpt,retrans,reject,notwait: Bool,init,k: Nat,to,from: adress,tmode: mmode)
   : exit(Nat,Nat,Nat,Nat,pqueue,pqueue,pacqueue,Bool,Bool,Bool,Bool,Bool,Bool,Nat,Bool,Nat,
        frmrreason,Nat,Bit) :=
      [not(lbusy)] ->
          (((( phl ! Input ? f: eframe[((pf(f) eq 0 of Bit) and (correct(f) and
                (is_i(get_f(f)) and (ns(f) eq (vr mod k) of Nat))))]
              ;exit(nr(f),qdlr,f,q) (* no poll requested *)
         |[phl]|
            validnrf[phl](vs,lu,k,q)
           ) >> accept nrcv: Nat,qdlr: pacqueue,f: eframe,q: pqueue in
                exit(nrcv,qdlr,f,q,false,false)
            )
         []
          (( phl ! Input ? f: eframe[((pf(f) eq 1 of Bit) and (((correct(f)) and ((is_i(get_f(f)))
                                   and (ns(f) eq (vr mod k) of Nat)))))]
           ;exit(nr(f),qdlr,f,q)
           |[phl]|
           validnrf[phl](vs,lu,k,q)
          ) >> accept nrcv: Nat,qdlr: pacqueue,f: eframe,q: pqueue in
         ((phl ! Out ! m(make_rr(flagging,from,ctl_sframe(1,((vr + Succ(0)) mod k)),fc,flagging),
                       noinfor,correct,notless,noabort)
               ;exit(nrcv,qdlr,f,q,true of Bool,false)
              )
             [] (* local entity becomes busy *)
          (phl ! Out ! m(make_rnr(flagging,from,ctl_sframe(1,((vr + Succ(0)) mod k)),fc,flagging),
                       noinfor,correct,notless,noabort)
               ;exit(nrcv,qdlr,f,q,true of Bool,true of Bool)
              ))
          ))
         >> accept nrcv: Nat,qdlr: pacqueue,f: eframe,q: pqueue,chk: Bool,lbusy: Bool in
            ([not(isempty(q)) and (nrcv ne lu)] ->
                     (* receive nr higher than last received nr acking some i frames *)
               ( t ! reset              (* stop timer *)
                ;releasequeue[phl] (k,nrcv,q)
                >> accept lu: Nat, q: pqueue in
                    exit(lu,vr + Succ(0),q,false,pack(f) +-- qdlr)
                   >> accept lu: Nat,vr: Nat,q: pqueue,starttime: Bool,qdlr: pacqueue in
                      ([isempty(q)] ->
                         (* if no outstanding frames unacked do not start timer again *)
                          ((([not(isempty(qdls))] ->
                           ([not(chk)] ->
                               (i;sendi[phl,t] (vs,vr,lu,vsp,q,qdls,qdlr,starttime,rbusy,lbusy,
                    chkpt,retrans,reject,notwait,init,k,to,from,tmode)
         []
         i;sendrr[phl] (vs,vr,lu,vsp,q,qdls,qdlr,starttime,rbusy,lbusy,chkpt,
                    retrans,reject,notwait,init,k,to,from,tmode)
         []
         i;sendrnr[phl] (vs,vr,lu,vsp,q,qdls,qdlr,starttime,rbusy,lbusy,chkpt,
        )
       []
       [chk] ->
        (exit(vs,vr,lu,vsp,q,qdls,qdlr,starttime,rbusy,lbusy,chkpt,retrans,
                reject,init,notwait,0 of Nat,nors,0 of Nat,0 of Bit))
    )
      []
      [isempty(qdls)] ->  (* dte/dce becomes busy *)
       ([not(chk)] ->
        (i;sendrnr[phl] (vs,vr,lu,vsp,q,qdls,qdlr,starttime,rbusy,lbusy,chkpt,
        []
         i;sendrr[phl] (vs,vr,lu,vsp,q,qdls,qdlr,starttime,rbusy,lbusy,chkpt,
                retrans,reject,notwait,init,k,to,from,tmode)
         )
      []
      [chk] ->
       (exit(vs,vr,lu,vsp,q,qdls,qdlr,starttime,rbusy,lbusy,chkpt,retrans,
                reject,init,notwait,0 of Nat,nors,0 of Nat,0 of Bit))
    )
   )
) >> accept vs: Nat,vr: Nat,lu: Nat,vsp: Nat,q: pqueue,qdls: pqueue,
        qdlr: pacqueue,starttime: Bool,rbusy: Bool,lbusy: Bool,chkpt: Bool,
        retrans: Bool,reject: Bool,init: Nat,notwait: Bool,typ: Nat,
        rs: frmrreason,a: Nat, b: Bit in
 (exit(vs,vr,lu,vsp,q,qdls,qdlr,true of Bool,rbusy,lbusy,chkpt,retrans,
        reject,init,notwait,typ,nors,0 of Nat,0 of Bit))
 )
[]
[not(isempty(q))]   -> (* stiill unacked frames start timer again  *)
 (( ((([not(isempty(qdls))] ->
     ([not(chk)] ->
      (i;sendi[phl,t] (vs,vr,lu,vsp,q,qdls,qdlr,starttime,rbusy,lbusy,chkpt,
                retrans,reject,notwait,init,k,to,from,tmode)
        []
        i;sendrr[phl] (vs,vr,lu,vsp,q,qdls,qdlr,starttime,rbusy,lbusy,chkpt,
                retrans,reject,notwait,init,k,to,from,tmode)
        []
        i;sendrnr[phl] (vs,vr,lu,vsp,q,qdls,qdlr,starttime,rbusy,lbusy,chkpt,
                retrans,reject,notwait,init,k,to,from,tmode)
        )
        []
        [chk] ->
        (exit(vs,vr,lu,vsp,q,qdls,qdlr,starttime,rbusy, lbusy,chkpt,retrans,
          reject,init,notwait,(Succ(0) of Nat),nors,0 of Nat,0 of Bit))
       )
      []
       [isempty(qdls)] ->
        ([not(chk)] ->
                        (i;sendrnr[phl] (vs,vr,lu,vsp,q,qdls,qdlr,starttime,rbusy,lbusy,
                                chkpt,retrans,reject,notwait,init,k,to,from,tmode)
                        []
                         i;sendrr[phl] (vs,vr,lu,vsp,q,qdls,qdlr,starttime,rbusy,lbusy,
                                chkpt,retrans,reject,notwait,init,k,to,from,tmode)
                        )
                        []
                      [chk] ->
                       (exit(vs,vr,lu,vsp,q,qdls,qdlr,starttime,rbusy,lbusy,chkpt,retrans,
                        reject,init,notwait,(Succ(0) of Nat),nors,0 of Nat,0 of Bit))
              )) )) )
              >> accept vs: Nat,vr: Nat,lu: Nat,vsp: Nat,q : pqueue,qdls: pqueue,
                        qdlr: pacqueue,starttime: Bool,rbusy: Bool,lbusy: Bool,chkpt: Bool,
                        retrans: Bool,reject: Bool,init: Nat,notwait: Bool,typ: Nat,
                        rs: frmrreason,ind: Nat,b: Bit in
                (exit(vs,vr,lu,vsp,q,qdls,qdlr,true of Bool,rbusy,lbusy,chkpt,retrans,
                        reject,init,notwait,typ,nors,0 of Nat,0 of Bit))
     )
    ))
    []
    [(isempty(q) or (nrcv eq lu))] ->
                (* nr received equal to last nr received *)
          (exit(lu,vr + Succ(0),q,true of Bool,pack(f) +-- qdlr)
      >> accept lu: Nat,vr: Nat,q: pqueue,starttime: Bool,qdlr: pacqueue in
          ((( ([not(isempty(qdls))] ->
                ([not(chk)] ->
                  (i;sendi[phl,t] (vs,vr,lu,vsp,q,qdls,qdlr,starttime,rbusy,lbusy,chkpt,
                          retrans,reject,notwait,init,k,to,from,tmode)
                  []
                  i;sendrr[phl] (vs,vr,lu,vsp,q,qdls,qdlr,starttime,rbusy,lbusy,chkpt,
                          retrans,reject,notwait,init,k,to,from,tmode)
                  []
                  i;sendrnr[phl] (vs,vr,lu,vsp,q,qdls,qdlr,starttime,rbusy,lbusy,chkpt,
                          retrans,reject,notwait,init,k,to,from,tmode)
                  )
                []
                [chk] ->
                  (exit(vs,vr,lu,vsp,q,qdls,qdlr,true of Bool,rbusy,lbusy,chkpt,retrans,
              )
            []
            [isempty(qdls)] ->
            (* dte/dce becomes busy or send rr since no info frame available *)
              ([not(chk)] ->
                (i;sendrnr[phl] (vs,vr,lu,vsp,q,qdls,qdlr,starttime,rbusy,lbusy,chkpt,
                          retrans,reject,notwait,init,k,to,from,tmode)
                  []
                  i;sendrr[phl] (vs,vr,lu,vsp,q,qdls,qdlr,starttime,rbusy,lbusy,chkpt,
                          retrans,reject,notwait,init,k,to,from,tmode)
                )
                []
                [chk] ->
                 (exit(vs,vr,lu,vsp,q,qdls,qdlr,true of Bool,rbusy,lbusy,chkpt,retrans,
                          reject,init,notwait,0 of Nat,nors,0 of Nat,0 of Bit))
              )
```

```
                )
                 ) )
                  >> accept vs: Nat,vr: Nat,lu: Nat,vsp: Nat,q: pqueue,qdls: pqueue,
                           qdlr: pacqueue,starttime: Bool,rbusy: Bool,lbusy: Bool,chkpt: Bool,
                           retrans: Bool,reject: Bool,init: Nat,notwait: Bool,typ: Nat,
                           rs: frmrreason,ind: Nat,b: Bit  in
                    (exit(vs,vr,lu,vsp,q,qdls,qdlr,true of Bool,rbusy,lbusy,chkpt,retrans,
                           reject,init,notwait,typ,nors,0 of Nat,0 of Bit))
        ))
      ))
    endproc (* receivevalidi  *)

   process rcviandbusy[phl,t] (vs,vr,lu,vsp: Nat,q,qdls: pqueue,qdlr: pacqueue,starttime,rbusy,
      lbusy,chkpt,retrans,reject,notwait: Bool,init,k: Nat,to,from: adress,tmode: mmode)
   : exit(Nat,Nat,Nat,Nat,pqueue,pqueue,pacqueue,Bool,Bool,Bool,Bool,Bool,Bool,Nat,Bool,
              Nat,frmrreason,Nat,Bit) :=
  [lbusy] ->
    ((phl ! Input ? f: eframe[(correct(f) and is_i(get_f(f))) and  ((pf(f) ne 1 of Bit)
                           and (ns(f) eq (vr mod k)))]
     ;exit(nr(f),qdlr,f,q)
     |[phl]|
     validnrf[phl](vs,lu,k,q)
     )
    []
    (( phl ! Input ? f: eframe[(correct(f) and is_i(get_f(f))) and  ((pf(f) eq 1 of Bit)
                           and (not(ns(f) eq (vr mod k))))]
    ;exit(nr(f),qdlr,f,q)
     |[phl]|
     validnrf[phl](vs,lu,k,q)
    ) >> accept nrcv: Nat,qdlr: pacqueue,f: eframe,q: pqueue in
            phl ! Out ! m(make_rnr(flagging,from,ctl_sframe(1,(vr mod k)),fc,flagging),
                           noinfor,correct,notless,noabort)
                 ;exit(nrcv,qdlr,f,q)
            )
    []
    (( phl ! Input ? f: eframe[(correct(f) and is_i(get_f(f))) and  ((pf(f) eq 1 of Bit)
                           and (ns(f) eq (vr mod k)))]
    ;exit(nr(f),qdlr,f,q)
     |[phl]|
     validnrf[phl](vs,lu,k,q)
    )
            >> accept nrcv: Nat,qdlr: pacqueue,f: eframe,q: pqueue in
                 phl ! Out ! m(make_rnr(flagging,from,ctl_sframe(1,(vr mod k)),fc,flagging),
                     noinfor,correct,notless,noabort)
                 ;exit(nrcv,qdlr,f,q)
            )
             )
      >> accept nrcv: Nat,qdlr: pacqueue,f : eframe,q: pqueue in
         (   [not(isempty(q)) and (nrcv ne lu)] ->
                  (* receive nr higher than last received nr acking some i frames *)
            ( t ! reset              (* stop timer *)
             ;releasequeue[phl] (k,nrcv,q)
             >> accept lu: Nat, q: pqueue in
                  ([isempty(q)] ->
                         (exit(vs,vr,lu,vsp,q,qdls,qdlr,starttime,rbusy,lbusy,chkpt,retrans,
                                 reject,init,notwait,0 of Nat,nors,0 of Nat,0 of Bit))
                         )
                       []
                       [not(isempty(q))] ->
                          (exit(vs,vr,lu,vsp,q,qdls,qdlr,starttime,rbusy,lbusy,chkpt,retrans,
                                 reject,init,notwait,(Succ(0) of Nat),nors,0 of Nat,0 of Bit))
                      )
             []
             [isempty(q) or (nrcv eq lu)] -> (* nr received equal to last nr received *)
                (exit(vs,vr,lu,vsp,q,qdls,qdlr,starttime,rbusy,lbusy,chkpt,retrans,reject,
                         init,notwait,0 of Nat,nors,0 of Nat,0 of Bit))
      )
      endproc      (* rcviandbusy *)

    process receiverr[phl,t] (vs,vr,lu,vsp: Nat,q,qdls: pqueue,qdlr: pacqueue,starttime,rbusy,
        lbusy,chkpt,retrans,reject,notwait: Bool,init,k: Nat,to,from: adress,tmode: mmode)
     : exit(Nat,Nat,Nat,Nat,pqueue,pqueue,pacqueue,Bool,Bool,Bool,Bool,Bool,Bool,Nat,Bool,
        Nat,frmrreason,Nat,Bit)  :=
        ((((phl ! Input ? f: eframe[is_rr(get_f(f)) and ((pf(f) eq 0 of Bit))]
         ;exit(nr(f),qdlr,f,q)             (* no poll requested from peer entity *)
         |[phl]|
         validnrf[phl](vs,lu,k,q)
         ) >> accept nrcv: Nat,qdlr: pacqueue,f: eframe,q: pqueue in
              exit(nrcv,qdlr,f,q,false,chkpt,init)
         )
         []
          ( (phl ! Input ? f: eframe[chkpt and (((pf(f) eq 1 of Bit) and (eq(adrs(f),to))) and
                                     (correct(f) and (is_rr(get_f(f)))))]
         ;exit(nr(f),qdlr,f,q) (* no poll requested *)
           |[phl]|
           validnrf[phl](vs,lu,k,q)
           ) >> accept nrcv: Nat,qdlr: pacqueue,f: eframe,q: pqueue in
                exit(nrcv,qdlr,f,q,true of Bool,false,0 of Nat)
           )
         []
         ((phl ! Input ? f: eframe[is_rr(get_f(f)) and ((pf(f) eq 1 of Bit) and (eq(adrs(f),from)))]
          ;exit(nr(f),qdlr,f,q)
          |[phl]|
          validnrf[phl](vs,lu,k,q)
           ) >> accept nrcv: Nat,qdlr: pacqueue,f: eframe ,q: pqueue
in
       ([not(lbusy)] ->       (* local entitty not busy *)
            ([not(reject)] ->
                ( phl ! Out ! m(make_rr(flagging,from,ctl_sframe(1,(vr mod k)),fc,flagging),
                                        noinfor,correct,notless,noabort)
                 ;exit(nrcv,qdlr,f,q,false,chkpt,init) (* poll requested so rr with f=1 sent *)
                )
             []
             [reject] ->
              (( phl ! Out ! m(make_rr(flagging,from,ctl_sframe(1,(vr mod k)),fc,flagging),
                                noinfor,correct,notless,noabort)
                 ;exit(nrcv,qdlr,f,q,false,chkpt,init)
               )
        []
        ( phl ! Out ! m(make_rej(flagging,from,ctl_sframe(1,(vr mod k)),fc,flagging),noinfor,
          ;exit(nrcv,qdlr,f,q,false,chkpt,init)
       )
     ))
      []
      [lbusy] ->             (* local entity busy send rnrn *)
         phl ! Out ! m(make_rnr(flagging,from,ctl_sframe(1,(vr mod k)),fc,flagging),noinfor,
                                correct,notless,noabort)
           ;exit(nrcv,qdlr,f,q,false,chkpt,init)
     )
    ))
  >> accept nrcv: Nat,qdlr: pacqueue,f: eframe,q: pqueue,retranschk: Bool,chk: Bool,
                 init: Nat in
     ([(not(isempty(q)) and (nrcv ne lu)) and not(retranschk)] ->
              (* receive nr higher than last received nr acking some i frames *)
        ( t ! reset               (* stop timer *)
         ;releasequeue[phl] (k,nrcv,q)
         >> accept lu: Nat, q: pqueue in
             exit(lu,false,q,qdls,qdlr,retranschk,chk)
        )
     []
     [(isempty(q) or (nrcv eq lu)) or retranschk] ->
                    (* nr received equal to last nr received     *)
           exit(lu,true of Bool,q,qdls,qdlr,retranschk,chk)
     ) >> accept lu: Nat,fin: Bool,q: pqueue,qdls: pqueue,qdlr: pacqueue,retranschk: Bool,
                 chkpt: Bool in
     ([(not(retrans) and not(retranschk)) and fin] ->
       (exit(vs,vr,lu,vsp,q,qdls,qdlr,true of Bool,rbusy,lbusy,chkpt,retrans,reject,init,
                 notwait,0 of Nat,nors,0 of Nat,0 of Bit))
      []
      [(not(retrans) and not(retranschk)) and (not(fin) and isempty(q))] ->
       (exit(vs,vr,lu,vsp,q,qdls,qdlr,starttime,rbusy,lbusy,chkpt,retrans,reject,init,
                 notwait,0 of Nat,nors,0 of Nat,0 of Bit))
      []
      [(not(retrans) and not(retranschk)) and (not(fin) and not(isempty(q)))]  ->
       (exit(vs,vr,lu,vsp,q,qdls,qdlr,starttime,rbusy,lbusy,chkpt,retrans,reject,init,
                 notwait,(Succ(0) of Nat),nors,0 of Nat,0 of Bit))
      []
      [retrans or retranschk] ->
        ([fin or retranschk] ->
             (hide t in
               retransmit[phl,t] (vs,vr,lu,vsp,q,qdls,qdlr,starttime,rbusy,lbusy,chkpt,
                                  retrans,reject,notwait,init,k,to,from,tmode)
             )
         []
         [not(fin) and not(retranschk)] ->
             (hide t in
             (t ! start
              ;retransmit[phl,t] (vs,vr,lu,vsp,q,qdls,qdlr,starttime,rbusy,lbusy,chkpt,
                                  retrans,reject,notwait,init,k,to,from,tmode)
             ))
        )
    ))
```

```
endproc  (* receiverr  *)

process retransmit[phl,t] (vs,vr,lu,vsp: Nat,q,qdls: pqueue,qdlr: pacqueue,starttime,rbusy,
    lbusy,chkpt,retrans,reject,notwait: Bool,init,k: Nat,to,from: adress,tmode: mmode)
 : exit(Nat,Nat,Nat,Nat,pqueue,pqueue,pacqueue,Bool,Bool,Bool,Bool,Bool,Bool,Nat,Bool,
    Nat,frmrreason,Nat,Bit) :=
   retransmission[phl] (vs,vr,lu,vsp,q,q,qdls,qdlr,starttime,rbusy,lbusy,chkpt,retrans,
                         reject,notwait,init,k,to,from,tmode)
   [> (receivernr[phl,t] (vs,vr,lu,vsp,q,qdls,qdlr,starttime,rbusy,lbusy,chkpt,retrans,
                         reject,notwait,init,k,to,from,tmode))
endproc (* retransmit *)

process sendrr[phl] (vs,vr,lu,vsp: Nat,q,qdls: pqueue,qdlr: pacqueue,starttime,rbusy,lbusy,
    chkpt,retrans,reject,notwait: Bool,init,k: Nat,to,from: adress,tmode: mmode)
 : exit(Nat,Nat,Nat,Nat,pqueue,pqueue,pacqueue,Bool,Bool,Bool,Bool,Bool,Bool,Nat,Bool,
    Nat,frmrreason,Nat,Bit) :=
 [not(rbusy)] ->
  (((i;phl ! Out ! m(make_rr(flagging,to,ctl_sframe(0,(vr mod k)),fc,flagging),noinfor,
           correct,notless,noabort)
    ;exit(vs,vr,lu,vsp,q,qdls,qdlr,starttime,rbusy,false,chkpt,retrans,reject,init,notwait,
           0 of Nat,nors,0 of Nat,0 of Bit)
 )
 []
 (i;phl ! Out ! m(make_rr(flagging,from,ctl_sframe(0,(vr mod k)),fc,flagging),noinfor,
           correct,notless,noabort)
   ;exit(vs,vr,lu,vsp,q,qdls,qdlr,starttime,rbusy,false,chkpt,retrans,reject,init,notwait,
           0 of Nat,nors,0 of Nat,0 of Bit)
 )
 []
  (i;phl ! Out ! m(make_rr(flagging,to,ctl_sframe(1,(vr mod k)),fc,flagging),noinfor,correct,
                   notless,noabort)
  ;exit(vs,vr,lu,vsp,q,qdls,qdlr,starttime,rbusy,false,true of Bool,retrans,reject,init,
           notwait,0 of Nat,nors,0 of Nat,0 of Bit)
   ))
  []
   i;exit(vs,vr,lu,vsp,q,qdls,qdlr,starttime,rbusy,lbusy,chkpt,retrans,reject,init,notwait,
           0 of Nat,nors,0 of Nat,0 of Bit))
endproc

process sendrej[phl] (vs,vr,lu,vsp: Nat,q,qdls: pqueue,qdlr: pacqueue,starttime,rbusy,
    lbusy,chkpt,retrans,reject,notwait: Bool,init,k: Nat,to,from: adress,tmode: mmode)
 : exit(Nat,Nat,Nat,Nat,pqueue,pqueue,pacqueue,Bool,Bool,Bool,Bool,Bool,Bool,Nat,Bool,Nat,
    frmrreason,Nat,Bit) :=
   ([not(lbusy) and not(reject)] ->
    (((( phl ! Input ? f: eframe[(correct(f) and is_i(get_f(f))) and ((ns(f) ne (vr mod k)))]
      ;exit(nr(f),qdlr,f,q)
  |[phl]|
      validnrf[phl](vs,lu,k,q)
     )
    ) >> accept nrcv: Nat,qdlr: pacqueue,f: eframe,q: pqueue in
           releasequeue[phl](k,nrcv,q)
     >> accept lu: Nat,q: pqueue in
        ((i;phl ! Out ! m(make_rej(flagging,to,ctl_sframe(0,(vr mod k)),fc,flagging),noinfor,
           correct,notless,noabort)
```

```
          ;exit(lu,vr,q,qdls,qdlr,false)
         )
        []
        (i;phl ! Out ! m(make_rej(flagging,to,ctl_sframe(1,(vr mod k)),fc,flagging),noinfor,
                     correct,notless,noabort)
         ;exit(lu,vr,q,qdls,qdlr,true of Bool))
         []
         (i;phl ! Out ! m(make_rej(flagging,from,ctl_sframe(0,(vr mod k)),fc,flagging),noinfor,
                     correct,notless,noabort)
         ;exit(lu,vr,q,qdls,qdlr,false)
        ))))
       []
       [reject] ->
          (((( phl ! Input ? f: eframe[((correct(f) and is_i(get_f(f))) and
                                   ((ns(f) ne (vr mod k)))) and (pf(f) eq 0 of Bit)]
       ;exit(nr(f),qdlr,f,q)
   |[phl]|
       validnrf[phl](vs,lu,k,q)
      )
       >> accept nrcv: Nat,qdlr: pacqueue,f: eframe,q: pqueue in
       releasequeue[phl](k,nrcv,q)
          )
          []
          ((phl ! Input ? f: eframe[((correct(f) and is_i(get_f(f))) and
                                   ((ns(f) ne (vr mod k)))) and (pf(f) eq 1 of Bit)]
       ;exit(nr(f),qdlr,f,q)
   |[phl]|
       validnrf[phl](vs,lu,k,q)
          )
        >> accept nrcv: Nat,qdlr: pacqueue,f: eframe,q: pqueue in

          phl ! Out ! m(make_rej(flagging,from,ctl_sframe(1,(vr mod k)),fc,flagging),noinfor,
                                   correct,notless,noabort)
       ;releasequeue[phl](k,nrcv,q)
         ))
       >> accept lu: Nat,q: pqueue in
          (exit(lu,vr,q,qdls,qdlr,chkpt)))
          )
    >> accept lu: Nat,vr: Nat,q: pqueue,qdls: pqueue,qdlr: pacqueue,chkpt: Bool in
         ([isempty(q)]  ->
              (exit(vs,vr,lu,vsp,q,qdls,qdlr,true of Bool,rbusy,lbusy,chkpt,retrans,reject,
                     init,notwait,0 of Nat,nors,0 of Nat,0 of Bit))
          []
          [not(isempty(q))] ->
             exit(vs,vr,lu,vsp,q,qdls,qdlr,true of Bool,rbusy,lbusy,chkpt,retrans,reject,
                     init,notwait,(Succ(0) of Nat),nors,0 of Nat,0 of Bit))
endproc (* sendrej *)

process receiverej[phl] (vs,vr,lu,vsp: Nat,q,qdls: pqueue,qdlr: pacqueue,starttime,rbusy,lbusy,
    chkpt,retrans,reject,notwait: Bool,init,k: Nat,to,from: adress,tmode: mmode)
 : exit(Nat,Nat,Nat,Nat,pqueue,pqueue,pacqueue,Bool,Bool,Bool,Bool,Bool,Bool,Nat,Bool,Nat,
    frmrreason,Nat,Bit) :=
    ((((phl ! Input ? f: eframe[is_rej(get_f(f)) and ((pf(f) eq 0 of Bit))]
     ;exit(nr(f),qdlr,f,q)             (* no poll requested from peer entity *)
```

```
        |[phl]|
        validnrf[phl](vs,lu,k,q)
        ) >> accept nrcv: Nat,qdlr: pacqueue,f: eframe,q: pqueue in
             exit(nrcv,qdlr,f,q,false,chkpt,init)
        )
        []
         ( (phl ! Input ? f: eframe[chkpt and (((pf(f) eq 1 of Bit) and (eq(adrs(f), to))) and
                                      (correct(f) and (is_rej(get_f(f)))))]
        ;exit(nr(f),qdlr,f,q) (* no poll requested *)
          |[phl]|
          validnrf[phl](vs,lu,k,q)
          ) >> accept nrcv: Nat,qdlr: pacqueue,f: eframe,q: pqueue in
               exit(nrcv,qdlr,f,q,true of Bool,false,0 of Nat)
          )
        []
        ((phl ! Input ? f: eframe[is_rej(get_f(f)) and ((pf(f) eq 1 of Bit) and (eq(adrs(f),from)))]
         ;exit(nr(f),qdlr,f,q)
         |[phl]|
         validnrf[phl](vs,lu,k,q)
          ) >> accept nrcv : Nat,qdlr: pacqueue,f: eframe ,q: pqueue in
           ([not(lbusy)] ->       (* local entitty not busy *)
              ([not(reject)] ->
                 ( phl ! Out ! m(make_rr(flagging,from,ctl_sframe(1,(vr mod k)),fc flagging),
                                      noinfor,correct,notless,noabort)
                ;exit(nrcv,qdlr,f,q,false,chkpt,init) (* poll requested so rr with f=1 sent *)
                 )
               []
               [reject] ->
                (( phl ! Out ! m(make_rr(flagging,from,ctl_sframe(1,(vr mod k)),fc,flagging),
                                      noinfor,correct,notless,noabort)
                ;exit(nrcv,qdlr,f,q,false,chkpt,init) (* poll requested so rr with f=1 sent *)
                 )
                 []
                 ( phl ! Out ! m(make_rej(flagging,from,ctl_sframe(1,(vr mod k)),fc,flagging),
                                      noinfor,correct,notless,noabort)
                ;exit(nrcv,qdlr,f,q,false,chkpt,init)
                 )))
            []
            [lbusy] ->             (* local entity busy send rnrn *)
               phl ! Out ! m(make_rnr(flagging,from,ctl_sframe(1,(vr mod k)),fc,flagging),
                              noinfor,correct,notless,noabort)
                 ;exit(nrcv,qdlr,f,q,false,chkpt,init)
          )
         ))
        >> accept nrcv: Nat,qdlr: pacqueue,f: eframe,q: pqueue,retranschk: Bool,chk: Bool,
               init: Nat in
           releasequeue[phl](k,nrcv,q)
            >> accept lu: Nat,q: pqueue in
              (hide t in
               retransmit[phl,t] (vs,vr,lu,vsp,q,qdls,qdlr,starttime,rbusy,lbusy,chkpt,retrans,
                              reject,notwait,init,k,to,from,tmode)
           ))
    endproc
```

```
process sendrnr[phl] (vs,vr,lu,vsp: Nat,q,qdls: pqueue,qdlr: pacqueue,starttime,rbusy,
    lbusy,chkpt,retrans,reject,notwait : Bool,init,k: Nat,to,from: adress,tmode: mmode)
: exit (Nat,Nat,Nat,Nat,pqueue,pqueue,pacqueue,Bool,Bool,Bool,Bool,Bool,Bool,Nat,Bool,Nat,
    frmrreason,Nat,Bit) :=
  [(not(rbusy))] ->
    (((i;phl ! Out ! m(make_rnr(flagging,to,ctl_sframe(0,(vr mod k)),fc,flagging),noinfor,
                    correct,notless,noabort)
      ;exit(vs,vr,lu,vsp,q,qdls,qdlr,starttime,rbusy,true of Bool,chkpt,retrans,reject,init,
            notwait,0 of Nat,nors,0 of Nat,0 of Bit)
   )
   []
    (i;phl ! Out ! m(make_rnr(flagging,from,ctl_sframe(0,(vr mod k)),fc,flagging),noinfor,
                    correct,notless,noabort)
      ;exit(vs,vr,lu,vsp,q,qdls,qdlr,starttime,rbusy,true of Bool,chkpt,retrans,reject,init,
                    notwait,0 of Nat,nors,0 of Nat,0 of Bit)
   )
   []
    i;phl ! Out ! m(make_rnr(flagging,to,ctl_sframe(1,(vr mod k)),fc,flagging),noinfor,
                    correct,notless,noabort)
    ;exit(vs,vr,lu,vsp,q,qdls,qdlr,starttime,rbusy,true of Bool,chkpt,retrans,reject,init,
                    notwait,0 of Nat,nors,0 of Nat,0 of Bit)
    ))
   endproc

process receivernr[phl,t] (vs,vr,lu,vsp: Nat,q,qdls: pqueue,qdlr: pacqueue,starttime,rbusy,
    lbusy,chkpt,retrans,reject,notwait: Bool,init,k: Nat,to,from: adress,tmode: mmode)
: exit(Nat,Nat,Nat,Nat,pqueue,pqueue,pacqueue,Bool,Bool,Bool,Bool,Bool,Bool,Nat,Bool,
            Nat,frmrreason,Nat,Bit)  :=
    (((( (phl ! Input ? f: eframe[((pf(f) eq 0 of Bit) and (correct(f)) and (is_rnr(get_f(f))))]
     ;exit(nr(f),qdlr,f,q) (* no poll requested *)
       |[phl]|
       validnrf[phl](vs,lu,k,q)
       ) >> accept nrcv: Nat,qdlr : pacqueue,f: eframe,q: pqueue in
        exit(nrcv,qdlr,f,q,false,chkpt,init)
       )
     []
     ( (phl ! Input ? f: eframe[(chkpt and (((pf(f) eq 1 of Bit) and (eq(adrs(f),to)))
                                   and (correct(f) and (is_rnr(get_f(f))))))]
     ;exit(nr(f),qdlr,f,q) (* no poll requested *)
       |[phl]|
       validnrf[phl](vs,lu,k,q)
       ) >> accept nrcv: Nat,qdlr: pacqueue,f: eframe,q: pqueue in
            exit(nrcv,qdlr,f,q,true of Bool,false,0 of Nat)
       )
     []
     (( (phl ! Input ? f: eframe[(((pf(f) eq 1 of Bit) and (eq(adrs(f),from)))
                           and (((correct(f)) and ((is_rnr(get_f(f)))))))]
       ;exit(nr(f),qdlr,f,q)
       |[phl]|
       validnrf[phl](vs,lu,k,q)
       )
       >> accept nrcv: Nat,qdlr: pacqueue,f: eframe,q: pqueue in
          ([not(lbusy)] ->
             ([not(reject)] ->
              ( phl ! Out ! m(make_rr(flagging,from,ctl_sframe(1,(vr mod k)),fc,flagging),
                          noinfor,correct,notless,noabort)
               ;exit(nrcv,qdlr,f,q,false,chkpt,init)
              )
             []
             [reject] ->
              ((i;phl ! Out ! m(make_rr(flagging,from,ctl_sframe(1,(vr mod k)),fc,flagging),
                          noinfor,correct,notless,noabort)
             ;exit(nrcv,qdlr,f,q,false,chkpt,init)
              )
             []
             (i;phl ! Out ! m(make_rej(flagging,from,ctl_sframe(1,(vr mod k)),fc,flagging),
                          noinfor,correct,notless,noabort)
             ;exit(nrcv,qdlr,f,q,false,chkpt,init)
              )
            )   )
           []
           [(lbusy)] ->
              phl ! Out ! m(make_rnr(flagging,from,ctl_sframe(1,(vr mod k)),fc,flagging),
                          noinfor,correct,notless,noabort)
            ;exit(nrcv,qdlr,f,q,false,chkpt,init)
          ))
        )
      ))  >> accept nrcv: Nat,qdlr: pacqueue,f: eframe,q: pqueue,retranschk: Bool,chk: Bool,
                   init: Nat in
       ([not(isempty(q)) and (nrcv ne lu)] ->
           (t ! reset
            ;releasequeue[phl](k,nrcv,q)
              >> accept lu: Nat,q: pqueue in
                  exit(lu,q,qdls,true of Bool,chk)  (* timer stopped *)
           )
        []
        [isempty(q) or (nrcv eq lu)] ->
           exit(lu,q,qdls,false,chk)    (* timer not stopped *)
       )
       )
       >> accept lu: Nat,q: pqueue,qdls: pqueue,stoptimer: Bool,chkpt: Bool in
         ([stoptimer and (isempty(qdls) and isempty(q))] ->
             (exit(vs,vr,lu,vsp,q,qdls,qdlr,false,rbusy,lbusy,chkpt,retrans,reject,init,
                     notwait,(Succ(Succ(0)) of Nat),nors,0 of Nat,0 of Bit))
          []
          [stoptimer and (not(isempty(qdls)) or not(isempty(q)))] ->
             (exit(vs,vr,lu,vsp,q,qdls,qdlr,true of Bool,rbusy,lbusy,chkpt,retrans,reject,init,
                     notwait,(Succ(Succ(Succ(0))) of Nat),nors,0 of Nat,0 of Bit))
          []
          [not(stoptimer) and (isempty(qdls) and isempty(q))] ->
             (exit(vs,vr,lu,vsp,q,qdls,qdlr,true of Bool,rbusy,lbusy,chkpt,retrans,reject,init,
                     notwait,0 of Nat,nors,0 of Nat,0 of Bit))
          []
          [not(stoptimer) and (not(isempty(qdls)) or not(isempty(q)))] ->
             (exit(vs,vr,lu,vsp,q,qdls,qdlr,true of Bool,rbusy,lbusy,chkpt,retrans,reject,init,
                     notwait,0 of Nat,nors,0 of Nat,0 of Bit))
         )
  endproc
```

```
process  timeout[phl,t] (n2,vs,vr,lu,vsp: Nat,q,qdls: pqueue,qdlr: pacqueue,starttime,rbusy,
    lbusy,chkpt,retrans,reject,notwait: Bool,init,k : Nat,to,from: adress,tmode: mmode)
 : exit(Nat,Nat,Nat,Nat,pqueue,pqueue,pacqueue,Bool,Bool,Bool,Bool,Bool,Bool,Nat,Bool,Nat,
    frmrreason,Nat,Bit) :=
    ([not(reject)] ->
      (t ! expired
      ;([init ne n2] ->
        (i;sendrrb[phl] (vs,vr,lu,vsp,q,qdls,qdlr,starttime,rbusy,lbusy,chkpt,retrans,reject,
                    notwait,init,k,to,from,tmode)
         []
         i;sendrnrb[phl] (vs,vr,lu,vsp,q,qdls,qdlr,starttime,rbusy,lbusy,chkpt,retrans,reject,
                    notwait,init,k,to,from,tmode)
        )
        []
        [init eq n2] ->
         (exit(vs,vr,lu,vsp,q,qdls,qdlr,true of Bool,rbusy,lbusy,chkpt,retrans,reject,init,
            notwait,Succ(Succ(Succ(Succ(0))) of Nat),make_reason(ctl_uframe(1),0 of Bit,
            0 of Nat,0 of Bit,0 of Nat,0 of Bit,0 of Bit, 0 of Bit,0 of Bit,0 of Bit),
            0 of Nat,1 of Bit ))
      ))
    []
    [reject] ->
       (t ! expired
       ;([init ne n2] ->
          (sendrejb[phl] (vs,vr,lu,vsp,q,qdls,qdlr,starttime,rbusy,lbusy,chkpt,retrans,reject,
                    notwait,init,k,to,from,tmode))
         []
         [init eq n2] ->
          (exit(vs,vr,lu,vsp,q,qdls,qdlr,true of Bool,rbusy,lbusy,chkpt,retrans,reject,init,
                    notwait,(Succ(Succ(Succ(Succ(0)))) of Nat),make_reason(ctl_uframe(1),
                    0 of Bit,0 of Nat,0 of Bit,0 of Nat,0 of Bit,0 of Bit, 0 of Bit,0 of Bit,
                    0 of Bit),0 of Nat,1 of Bit))
       ))
        )
    where
    process sendrrb[phl] (vs,vr,lu,vsp: Nat,q,qdls: pqueue,qdlr: pacqueue,starttime,rbusy,
            lbusy,chkpt,retrans,reject,notwait: Bool,init,k: Nat,to,from: adress,tmode: mmode)
    : exit(Nat,Nat,Nat,Nat,pqueue,pqueue,pacqueue,Bool,Bool,Bool,Bool,Bool,Bool,Nat,Bool,Nat,
            frmrreason,Nat,Bit) :=
         phl ! Out ! m(make_rr(flagging,to,ctl_sframe(1,(vr mod k)),fc,flagging),noinfor,
            correct,notless,noabort)
        ;(exit(vs,vr,lu,vsp,q,qdls,qdlr,true of Bool,rbusy,false,true of Bool,retrans,reject,
            init,notwait,(Succ(Succ(0)) of Nat),nors,0 of Nat,0 of Bit))
    endproc  (* sendrrb *)

    process sendrnrb[phl] (vs,vr,lu,vsp: Nat,q,qdls: pqueue,qdlr: pacqueue,starttime,rbusy,
            lbusy,chkpt,retrans,reject,notwait: Bool,init,k: Nat,to,from: adress,tmode: mmode)
    : exit(Nat,Nat,Nat,Nat,pqueue,pqueue,pacqueue,Bool,Bool,Bool,Bool,Bool,Bool,Nat,Bool,Nat,
            frmrreason,Nat,Bit) :=
         phl ! Out ! m(make_rnr(flagging,to,ctl_sframe(1,(vr mod k)),fc,flagging),noinfor,
            correct,notless,noabort)
        ;(exit(vs,vr,lu,vsp,q,qdls,qdlr,true of Bool,rbusy,true of Bool,true of Bool,retrans,
            reject,init,notwait,(Succ(Succ(0)) of Nat),nors,0 of Nat,0 of Bit))
        endproc  (* sendrrb *)

        process sendrejb[phl] (vs,vr,lu,vsp: Nat,q,qdls: pqueue,qdlr: pacqueue,starttime,rbusy,
                lbusy,chkpt,retrans,reject,notwait: Bool,init,k: Nat,to,from: adress,tmode: mmode)
        : exit(Nat,Nat,Nat,Nat,pqueue,pqueue,pacqueue,Bool,Bool,Bool,Bool,Bool,Bool,Nat,Bool,Nat,
                frmrreason,Nat,Bit) :=
            phl ! Out ! m(make_rej(flagging,to,ctl_sframe(1,(vr mod k)),fc,flagging),noinfor,
                correct,notless,noabort)
               ;(exit(vs,vr,lu,vsp,q,qdls,qdlr,true of Bool,rbusy,false,true of Bool,retrans,
                true of Bool,init,notwait,(Succ(Succ(0)) of Nat),nors,0 of Nat,0 of Bit))
        endproc  (* sendrejb *)

      endproc (*  timeoutbusy  *)

      endproc  (* infphase *)
     endproc (* inforphase *)
    endproc (* dataphase *)
endproc (* lapb *)

endspec
```