OSI formal specification case study: the Inres protocol and service, revised

Dieter Hogrefe
Institut für Informatik
Universität Bern
Länggassstrasse 51
CH-3012 Bern, Switzerland
May 1991
Update May 1992

Abstract
This paper contains an OSI specification case study. An informal specification of an OSI-like protocol and service is followed by an SDL [ZI80], Estelle [ISO 9074] and LOTOS [ISO 8807] specification of the same protocol and service. The protocol is called Inres, for Initiator-Responder protocol. It is connection oriented and asymmetric, i.e. one side can only establish connections and send data while the other side can accept connections, release them and receive data.

1. Introduction
The system under study, Inres, is not a real system, although it does contain many basic OSI concepts and is therefore very suitable for illustrative purposes because it is easy to understand and not too big. It is an abridged version of the Abracadabra system described in [TR 10167]. The Inres system has originally been published in [HOG89] in German and has already been used as a reference in many publications. This paper contains only a short evaluation and experience section at the end. The main purpose of the paper is to offer the community a well worked out protocol example, which has been checked in parts with tools to serve as:

- a reference for other work using the Inres protocol
- an illustration for the use of PDTs (formal description techniques)
- stimulate and provoke the discussion on protocol against service verification, automatic generation of conformance tests,
- stimulate and provoke experts of other formal description techniques such as Z [SP88], stream functions (BRO87), temporal logic (GOT91), to specify the same protocol with their approach.

In the following sections the services and the protocol are first described verbally and semi-formally with TS diagrams. These informal descriptions form the basis for the formal specifications with SDL.

There are some conventions in the descriptions for the naming of SPs, SAPs and SDUs. Those SPs, SAPs, and SDUs that are related to the Medium service have the prefix M. For example, MSDU is the name of a service data unit of the Medium service. SPs, SAPs, and SDUs that are related to the Inres service and protocol have the prefix I.

The order of the description in the next chapters is a recommended order: First, one should think about the service that has to be rendered, then the service that can be used is taken into account, and thereafter the protocol is designed which can render the desired service.

1.1 Informal specification of the Inres service
This is an abridged version of the Abracadabra service [TR 10167]. The service is connection-oriented. A user who wants to communicate with another user via the service must first initiate a connection before exchanging data. Fig. 1.2 shows the basic schema of the service with its SPs and SAPs.

\[
\begin{array}{c}
\text{Initiator user} \\
\text{ICONconf} \\
\text{IDISind} \\
\text{ISAPini} \\
\end{array}
\quad
\begin{array}{c}
\text{Responder user} \\
\text{ICONind} \\
\text{IDATind} \\
\text{IDISreq} \\
\end{array}
\]

Figure 1.2 The Inres service

For simplification purposes the service is not symmetrical. The service can be accessed on two SAPs. On the one SAP (the left one in Fig. 1.2) the Initiator-user can initiate a connection and afterwards send data. On the other SAP another user, Responder-user, can accept the connection or reject it. After acceptance it can receive data from the initiating user.

The following SPs are used for the communication between user and provider:

- ICONreq: request of a connection by Initiator-user
- ICONInd: indication of a connection by the provider
- ICONResp: response to a connection attempt by Responder-user
- ICONConf: confirmation of a connection by the provider
- IDATreq(SDUs): data from the Initiator-user to the provider, this SP has a parameter of type SDU
- IDATind(SDUs): data from the Provider to the Responder-user, this SP has a parameter of type SDU
- IDISreq: request of a disconnection by the Responder-user
- IDISind: indication of a disconnection by the provider

The order of SPs at the different SAPs is specified in Fig. 1.3a-1.3h with generalized TS-diagrams (see [TR 8509]).

\[
\begin{array}{c}
\text{ICONreq} \\
\end{array}
\quad
\begin{array}{c}
\text{ICONind} \\
\text{ICONResp} \\
\end{array}
\]

Figure 1.3a Successful connection establishment
1.2 Informal specification of the Medium service

The Medium service has two SAPs: MSAP1 and MSAP2. The service is symmetrical and operates connectionless. It can be accessed at the two SAPs by the SPs MDATreq and MDATind, both of which have a parameter of type MSDU.

With the SPs data (MSDUs) can be transmitted from one SAP to the other. The data transmission is unreliable, and data can be lost. But data cannot be corrupted or duplicated. Fig. 1.4 shows the overall schema of the service, and Fig. 1.5a-1.5b show the respective TS diagrams.

1.3 Informal specification of the Inres protocol
This section describes a protocol, which by use of the unreliable Medium service, renders the Inres service to users in the imaginary next higher layer. Fig. 1.6 shows the overall architecture of the protocol.

### General properties of the protocol

The Inres protocol is a connection-oriented protocol that operates between two protocol entities Initiator and Responder. The protocol entities communicate by exchange of the protocol data units CR, CC, DT, AK and DR. The meaning of the PDUs is specified below.

<table>
<thead>
<tr>
<th>PDU</th>
<th>Meaning</th>
<th>parameter</th>
<th>respective SPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR</td>
<td>connection establishment</td>
<td>none</td>
<td>ICONreq, ICONind</td>
</tr>
<tr>
<td>CC</td>
<td>connection confirmation</td>
<td>None</td>
<td>ICONresp, ICONconf</td>
</tr>
<tr>
<td>DT</td>
<td>data transfer</td>
<td>sequence number, ISDU</td>
<td>IDATreq, IDATind</td>
</tr>
<tr>
<td>AK</td>
<td>acknowledgement</td>
<td>sequence number</td>
<td>IDISreq, IDISind</td>
</tr>
<tr>
<td>DR</td>
<td>disconnection</td>
<td>none</td>
<td>IDISresp</td>
</tr>
</tbody>
</table>

The communication between the two protocol entities takes place in three distinct phases: the connection establishment phase, the data transmission phase, and the disconnection phase.

In each phase only certain PDUs and SPs are meaningful. Unexpected PDUs and SPs are ignored by the entities Initiator and Responder.

### Figure 1.6 The Inres protocol

#### Connection establishment phase

A connection establishment is initiated by the Initiator-user at the entity Initiator with an ICONreq. The entity Initiator then sends a CR to the entity Responder.

Responder answers with CC or DR. In the case of CC, Initiator issues an ICONconf to its user, and the data phase can be entered. If Initiator receives a DR from Responder, the disconnection phase is entered. If Initiator receives nothing at all within 5 seconds, CR is transmitted again. If, after 4 attempts, still nothing is received by Initiator, it enters the disconnection phase.

If Responder receives a CR from Initiator, the Responder-user gets an ICONind. The user can respond with ICONresp or IDISreq. ICONresp indicates the willingness to accept the connection. Responder thereafter sends a CC to Initiator, and the data transmission phase is entered. Upon receipt of an IDISreq, Responder enters the disconnection phase.

#### Data transmission phase

If the Initiator-user of the entity issues an IDATreq, the Initiator sends a DT to the Responder and is then ready to receive another IDATreq from the user. IDATreq has one parameter that is a service data unit ISDU, which is used by the user to transmit information to the peer user. This user data is transmitted transparently by the protocol entity Initiator as a parameter of the protocol data unit DT. After having sent a DT to Responder, Initiator waits for 5 seconds for a respective acknowledgement AK. Then the DT is sent again. After 4 unsuccessful transmissions, Initiator enters the disconnection phase.

DT and AK carry a one-bit sequence number (0 or 1) as a parameter. Initiator starts, after having entered the data transmission phase, with the transmission of a DT with sequence number 1. A correct acknowledgement of a DT has the
same sequence number. After receipt of a correct acknowledgement, the next DT with the next (i.e. other) sequence number can be sent. If Initiator receives an AK with incorrect sequence number, it sends the last DT once again. It is also sent again if the respective AK does not arrive within 5 seconds. A DT can only be sent 4 times. Afterwards Initiator enters the disconnection phase. The same happens upon receipt of a DR.

Following the establishment of a successful connection, Responder expects the first DT with the sequence number 1. After receipt of a DT with the expected number, Responder gives the ISDU as a parameter of an IDATInd to its user and sends an AK with the same sequence number to the Initiator. A DT with an unexpected sequence number is acknowledged with an AK with the sequence number of the last correctly received DT. The user data ISDU of an incorrect DT is ignored. If Responder receives a CR, it enters the connection establishment phase. And upon receipt of an IDISreq, it enters the disconnection phase.

Disconnection phase
An IDISreq from the Responder-user results in the sending of a DR by the Responder. Afterwards Responder can receive another connection establishment attempt CR from Initiator.

At the Initiator, the DR results in an IDISInd sent by the Initiator to its user. An IDISInd is also sent to the user after DT or CR have been sent unsuccessfully to the Responder. Then a new connection can be established.

2. Formal specification of Inres in SDL

At some places the formal specification has to add some information to that found in the informal one. This is because informal specifications tend to be incomplete: they sometimes leave things up to the intuition of the reader. Therefore, informal service and protocol specifications can interpreted correctly only if the reader has some universal knowledge about services and protocols. Examples are given in the following sections.

The basic approach to the specification of the services and protocol is as follows. We consider a system called Inres (shown in Example 2.1). The system contains exactly one block, the Inres_service. The processes of this block specify the behaviour of the service provider; one process for each service access point. In addition, the block has a substructure, which is the Inres_protocol (specified in Example 2.4). This protocol specification again contains a block for the specification of a service, the Medium service. This block can in turn have a substructure if a protocol has to be specified, which should render the Medium service. More on this approach can be found in [BHHT88]

The substructure specification is used in SDL to specify the behaviour of a block in more detail, as an alternative to a more abstract block specification in terms of interacting processes.

This approach to service and protocol specification takes two very basic aspects of OSI into account:

- First, that of the recursive nature of the OSI-IRM. A service can be defined by a protocol using the underlying service, which again can be defined by a protocol using the next lower underlying service, and so on. The recursion stops with the Physical Medium (see [ISO 7498]). This recursive definition is mapped on a repeated use of the substructure construct.

- Second, the very important aspect that the service can be seen as an abstraction of the protocol and the next lower service. This is expressed in SDL by an abstract "overview" block specification in terms of interacting processes.

2.1 The Inres service in SDL/GR

In Example 2.1 the service provider block Inres_service consists of two processes interconnected by a signal route. Each process models the behaviour of one service access point.

Example 2.1:  

```plaintext
NEWTYPE  /* insert type of service data unit here */
ENDNEWTYPE
/* Definition of macro "daemon" see "inres_protocol" */

SYSTEM

Inres_Service

SIGNAL

ICONconf, IDATreq(ISDUType), ICONreq, IDAT, ICONind, ICONresp, IDAT(ISDUType);

BLOCK

ISAP_Ini

SIGNAL

ICONreq, IDATreq(ISDUType), ICONconf, ICONind, ICONresp, IDAT(ISDUType);

ISAP_Resp

SIGNAL

ICONreq, IDATreq(ISDUType), ICONconf, ICONind, ICONresp, IDAT(ISDUType);

Example 2.2:  

In principle, it would have been possible to model the whole behaviour of the service by just one process. But the multi-process solution usually results in a less complex specification. Especially in situations in which difficult collision situations may occur (this is not the case here, but is, for example, in the Abracadabra protocol in [ISO 10167]), it is very useful to model each service access point separately.

Example 2.2 shows the behaviour of the Initiator-SAP called ISAP_Manager_Ini and Example 2.3 shows the behaviour of the Responder-SAP called ISAP_Manager_Res. ISAP_Manager_Ini and ISAP_Manager_Res can communicate through a channel to establish the global behaviour of the service.

Example 2.2:  

```
The SDL specification of the service relies on the TS diagrams of Section 1.1. Since the TS diagrams do not have a formal semantics, whereas SDL does, no one-to-one mapping between the diagrams and the SDL specification is possible. Some information has to be added for formal specification of the service.

The Inres service is connection-oriented. Therefore, we will distinguish between the three phases connection establishment, data transfer, and disconnection.

In the following, not all features of the SDL specification are discussed; rather, only those are commented on which may not be obvious to the reader.

**Connection establishment**

Fig. 1.3a-1.3e illustrate the basic behaviour of the service provider during the connection establishment phase. Fig. 1.3a and Fig. 1.3b show the "normal" course of events, first a successful connection establishment and second a user-rejected connection attempt. Fig. 1.3c-1.3e show unpredictable non-deterministic behaviour of the service provider. In Fig. 1.3c the service provider does not indicate the connection attempt to the Responder-user, and in Fig. 1.3d the response of the Responder-user is not transmitted to the Initiator-user. In Fig. 1.3e the Responder-user does not respond "in time."

The modelling of the "normal" course of events in SDL is quite obvious. The difficulties arise from the various "abnormal" situations.

After the provider has received an ICONreq by the Initiator-user, basically two things can happen: Either the provider rejects the connection attempt with an IDISind to the Initiator-user (Fig. 1.3c); or the provider indicates an ICONind to the Responder-user (Fig. 1.3a). The latter is modelled by the sending of an ICON from ISAP_Manager_Init to ISAP_Manager_Resp. The Responder-user may answer with an ICONresp or an IDISreq. According to Fig. 1.3d, even if an ICONresp is issued to the provider, it may not be able to transmit it to the Initiator-user. The Initiator-user then receives an IDISind instead.
The TS diagram in Fig. 1.3e specifies the situation in which the Responder-user does not react “in time” upon receipt of the ICOInd - or does not react at all. This is modelled in SDL by the use of the timer construct. After a certain unspecified time, ISAP_Manager_Ini aborts the connection attempt on its own.

If Responder-user issues the ICOInd after the time-out, this results in a “half-open connection.” Initiator-user “thinks” the connection has been aborted, whereas Responder-user “thinks” the connection exists. ISAP_Manager_Ini is in state disconnected and ISAP_Manager_Ini is in state connected. If Initiator-user now tries to open a connection by issuing an ICOInd, ISAP_Manager_Res receives an ICON, issues an ICOInd to the user, and proceeds to state wait. This specific behaviour is not clearly specified by the TS diagrams, but it follows directly if one makes a model of the provider.

Data transfer

If a connection has been established successfully, the Initiator-user may issue an IDATreq with a parameter d of type ISDU to the ISAP_Manager_Ini. According to Fig. 1.3f and 1.3g, two things may happen: Either the data are issued to the Responder-user as an IDATInd, or the Initiator-user receives an IDISInd. In Example 2.2 this is modelled by the use of the Daemon after receipt of the signal IDATreq in state connected.

It is important to note that, in case of a disconnection during data transfer, the process ISAP_Manager_Ini may be in state disconnected, whereas the process ISAP_Manager_Res is still in state connected. This situation is terminated when the Initiator user tries to open up another connection. ISAP_Manager_Res then goes to state wait from state connected.

Disconnection

An IDISreq may be issued by the Responder-user at any time. According to Fig. 1.3h and 1.3i, an IDISreq may or may not result in an IDISInd at the Initiator-user. This is modelled by the Daemon in Example 2.3. Should the IDIS not be transmitted, the system runs into a half-open connection: ISAP_Manager_Res is in state disconnected while ISAP_Manager_Ini is in state connected and still trying to send data. But upon the first receipt ISAP_Manager_Res then aborts the connection with an IDIS. This situation is also captured by the TS diagram 1.3g.

2.2 The Inres Service in SDL/PR

system Inres_Service;
signal
ICOInd, IDATreq( ISDUType), ICOConf, ICOInd, IDATrep, IDISInd, IDATind( ISDUType), ICON, ICONF, IDIS, IDAT( ISDUType);
newtype ISDUType
literals 0, 1
/* insert type of service data unit here */
endnewtype;
channel ISAPresp
from ISAP_Resp to env
with ICONInd, IDATInd;
from env to ISAP_Resp
with ICONresp, IDISreq;
endchannel ISAPresp;
channel Internal
from ISAP_Ini to ISAP_Resp
with ICON, IDAT;
from ISAP_Resp to ISAP_Ini
with ICONF, IDIS;
endchannel Internal;
channel ISAPini
from ISAP_Ini to env
with ICONInd, IDISind;
from env to ISAP_Ini
with ICONreq, IDATreq;
endchannel ISAPini;
process ISAP_Resp referenced;
endblock ISAP_Resp;
block ISAP_Ini referenced;
endblock ISAP_Ini;
process ISAP_Manager_Resp;
dcl
d ISDUType;
start;
nextstate Disconnected;
state Wait;
input ICONresp;
decision ANY:
(EITHER)
output ICONInd;
nextstate Connected;
(OR)
nextstate Connected;
enddecision;
state Disconnected;
input ICON;
output ICONInd;
nextstate Wait;
input IDAT( d);
output IDIS;
nextstate -;
state Connected;
input IDAT( d);
2.3 The Inres protocol and Medium service in SDL/GR

Example 2.4 shows the overall structure of the Inres protocol together with the underlying Medium service as a substructure diagram (referenced in the block diagram Inres_service in Example 2.1)

MACRODEFINITION Datatype definitions

NEWTYPE Sequencenumber LITERALS 0, 1;
OPERATORS succ: Sequencenumber -> Sequencenumber;
AXIOMS succ(0) == 1;
succ(1) == 0;
ENDNEWTYPE Sequencenumber;

NEWTYPE MSDUType STRUCT id IPDUType;
num Sequencenumber;
ENDNEWTYPE MSDUType;

NEWTYPE IPDUType STRUCT id CR, CC, DR, DT, AK;
ENDNEWTYPE IPDUType;

NEWTYPE IPDUType STRUCT id MSDUType;
ENDNEWTYPE IPDUType;

NEWTYPE Medium

Example 2.4:

MACRODEFINITION Datatype definitions
The specification consists of three basic parts, all three of which are modelled by blocks: the two protocol entities Station_Init and Station_Res, and the service provider Medium.

Each Station consists of two processes. The Codex processes model the interface to the next lower layer by transforming the PDUs produced by the other processes (Initiator and Responder) into the SDUs of the next lower layer, which are then passed down as parameters of SPs.

This chosen architecture of a protocol entity is a useful one for all sorts of different protocols. Many protocol specifications nowadays describe the behaviour of the processes similar to Initiator and Responder, and they assume that there is an (abstract) channel between them which can be used to transmit the PDUs directly. Of course, according to the OSI-BRM, this is not the case: The service of the next lower layer has to be used for this communication. Therefore, the PDUs have to be transformed by processes like Codex_Init and Codex_Res.
In the most general case the Coder processes may have additional duties. According to [ISO 7498] (more precisely Section 5.7.4 in [ISO 7498]) these processes may handle the connection setup and maintenance of the next lower layer. More on this topic is given in [BHS91].

The SDL specification of the Ingres protocol is rather obvious and needs no further comments. It follows rather naturally from the informal description, although, similar to the service, some additional information had to be provided. The verification of the SDL specification with respect to the informal description is left to the reader.
2.4 The Inres protocol and Medium service in SDL/PR

Example 2.7:

Example 2.8:

Example 2.9:

MACRODEFINITION MSAP_Manager 1(1)

2.4 The Inres protocol and Medium service in SDL/PR

system INRES;
signal
with CC, AK, DR;
from Coder_Resp to Responder
with CR, DT;
signal route ISAP
from Responder to env
with ICONind, IDATind;
from env to Responder
with ICONresp, IDISreq;

process Coder_Resp [1, 1] referenced;

process Responder [1, 1] referenced;
endblock Res_Station;

process Coder_Init;
dcl d ISDUType, Num Sequencenumber, Sdu MSDUType;
start;
nextstate Idle;
state Idle;
input CR;
task Sdu!id := CR;grs0 :output MDATreq( Sdu);nextstate Idle;
input DT( Num, d);
task Sdu!id := DT, Sdu!Num := Num, Sdu!Data := d;
join grs0;
input MDATind( Sdu);
decision Sdu!id;( CC) :
output CC;grs1 :nextstate Idle;
( AK) :
output AK( Sdu!Num);join grs1;
( DR) :
output DR;join grs1;
else :
nextstate Idle;
enddecision;
endprocess Coder_Init;

process Initiator;
dcl d Counter Integer, d ISDUType, Num, Sequencenumber;
timer T;
synonym P Duration = 5;
start;
nextstate Disconnected;
state Disconnected;
input ICONreq;
task Counter := 1;output CR;set( now + P, T);nextstate Wait;
input DR;
reset( T);task Number := 1;output IDISreq;
nextstate Connected;
input T;
decision Counter < 4;
| TRUE) :
output CR;
task Counter := Counter + 1;
set( now + P, T);nextstate Sending;
input DR;
output IDISind;
nexstate Disconnected;
| FALSE) :
output IDISind;
nexstate Disconnected;
enddecision;
input DR;
reset( T);output IDISind;
nexstate Disconnected;
state Connected;
input DT( Number, d);
task Counter := 1;
set( now + P, T);nextstate Sending;
input DR;
output IDISind;
nexstate Disconnected;
state Sending;
input T;
nextstate Connected;

process MSAP_Manager2;
dcl d MSDUTyp;
start;
nexstate Idle;
state Idle;
input MAATreq( d);
decision ANY;
| EITHER) :
nexstate Idle;
| OR) :
output IDAT( d);nexstate Idle;
enddecision;
input AK( Num);
reset( T);nextstate Disconnected;
save IDATreq;
endprocess Initiator;

process MSAP_Manager1;
dcl d MSDUTyp;
start;
nexstate Idle;
state Idle;
input MAATreq( d);
decision ANY;
| EITHER) :
nexstate Idle;
| OR) :
output IDAT( d);nexstate Idle;
enddecision;
input IDAT( d);
output MAATind( d);
nexstate Idle;
endprocess MSAP_Manager2;

process MSAP_Manager1;
dcl d MSDUTyp;
3. Formal specification of Inres in Estelle

3.1 The Inres service in Estelle

This section describes the Inres service in Estelle. Figure 3.1 gives an overview on the specification. It consists of two modules User plus the module Service_provider. The Service_provider itself consists of two modules Initiator and Responder which define the behaviour at the two service access points. They communicate via the channel INTERNchn. The specification is very similar to the SDL specification, therefore any comments made there also apply here.
This section describes the Inres protocol in Estelle. The basic structure of the specification is depicted in Figure 3.2. The specification is very similar to the SDL specification, therefore any comments made there also apply here.
This section describes the Inres service in LOTOS. The specification style is constraint oriented [VSS88]. Constraints specify parts of the total behaviour of a system which are combined via the parallel operator. In the following example there are three constraints which define the:

- behaviour at the service access point ISAPini (ICEPini)
- behaviour at the service access point ISAPres (ICEPRes)
- end-to-end behaviour related to the events at the service access points (EndtoEnd)

The sequences of events ICEPini, ICEPRes and EndtoEnd are first defined independently from each other. Then they are coordinated by the parallel operator to define the overall behaviour of the system.
where
  process ICEPini[g] : noexit :=
    ( ConnectionphaseIni[g];
      DataphaseIni[g];
    )
  DisconnectionIni[g]
where
  process ConnectionphaseIni[g] : exit :=
    g! ICONreq;
    g! ICONconf;
  endproc (* ConnectionphaseIni *)
  process DataphaseIni[g] : noexit :=
    g! IDATreq par:ISDU;
  endproc (* DataphaseIni *)
  process DisconnectionIni[g] : noexit :=
    g! IDISind;
    ICEPini[g];
  endproc (* DisconnectionIni *)
endproc (* ICEPini *)

process ICEPres[g] : noexit :=
  ( ConnectionphaseRes[g];
    DataphaseRes[g];
  )
endproc (* ICEPres *)

process EndtoEnd[ini,res] : noexit :=
  ( ConnectionphaseEte[ini,res];
    DataphaseEte[ini,res];
  )
endproc (* EndtoEnd *)

4.2 The Inres protocol and Medium service in LOTOS

This section describes the Inres protocol and Medium service. While the Inres service specification was constraint oriented, this specification is state oriented according to [VSS88]. Fig. 4.1 depicts the basic architecture of the example.
x ≤ y = (x < y) or (x = y);
>x y = not (x < y);
>< y = not (x ≤ y);

ofsort DecNumb
1 = s(0);
2 = s(s(0));
3 = s(s(s(0)));
4 = s(s(s(s(0))));
5 = s(s(s(s(s(0)))));
6 = s(s(s(s(s(s(0))))));
7 = s(s(s(s(s(s(s(0)))))));
8 = s(s(s(s(s(s(s(s(s(0)))))))));
9 = s(s(s(s(s(s(s(s(s(s(0))))))))));

eqns forall f: Sequencenumber, d : ISDU, ipdu : IPDU
ofsort DecNumb
map(CR) = 0;
map(CC) = 1;
map(DT(f,d)) = 2;
map(AK(f)) = 3;
map(DR) = 4;

ofsort ISDU
data(DT(f,d)) = d;

ofsort DecNumb
map(IDATreq) = 3;
map(IDATind) = 4;

ofsort Boolean
isCR(ipdu) = map(ipdu) == 0;
isCC(ipdu) = map(ipdu) == 1;

ofsort ISDU
data(Serial) = d;

ofsort Boolean
isIDISreq(ipdu) = map(ipdu) == 3;

ofsort ISDU
map(IDATreq) = 3;
map(IDATind) = 4;

endproc (* MediumSpType *)

behaviour
hide MSAP1,MSAP2 in
Station_Ini[ISAPini,MSAP1] | [MSAP1] | Medium[MSAP1,MSAP2] | [MSAP2] | Station_Res[MSAP2,ISAPres]
where
process Medium[MSAP1,MSAP2] : noexit :=
   Channel[MSAP1,MSAP2] ||| Channel[MSAP2,MSAP1]
where
   process Channel[a,b] : noexit :=
      a?d:MSP [isMDATreq(d)];
      (b!MDATind(d);Channel[a,b][]i;Channel[a,b])
endproc (* Channel *)
endproc (* Medium *)

process Station_Ini[ISAPini,IPdu_ini]
where
process Initiator[ISAP,IPdu] : noexit :=
   (Connectionphase[ISAP,IPdu] >> Dataphase[ISAP,IPdu] (succ(0))) [>Disconnection[ISAP,IPdu]
where
   process Channel[a,b] : noexit :=
      a?p : Channel[a,b][]i,Channel[a,b]
endproc (* Channel *)
endproc (* Medium *)

process Station_Ini[ISAPini,IPdu_ini] : noexit :=
   Initiator[ISAPini,IPdu_ini]
   | [IPdu_ini] | Coder[IPdu_ini,ISAPini]
where
process Channel[a,b] : noexit :=
   (Connectionphase[ISAP,IPdu] >> Dataphase[ISAP,IPdu] (succ(0))) [>Disconnection[ISAP,IPdu]
where
process Connectionphase[ISAP,IPdu] : noexit :=
   Connectrequest[ISAP,IPdu] >> accept z:DecNumb in Wait[ISAP,IPdu](z)
where

process Connectrequest [ISAP, IPdu] : not (DecNumb) :=
  [ISAP?sp:SP; [i:ICOMReq(sp)] => IPdu!ICOMconf; exit [i]]
  [not (i:ICOMReq(sp))] => Connectrequest [ISAP, IPdu]
(* User errors are ignored *)

(* DR is only accepted by process Disconnection *)
[IPdu?ipdu:IPDU[not (i:ICOMReq(sp))]; Connectrequest [ISAP, IPdu]]
(* System errors are ignored *)
endproc (* Connectrequest *)

process Wait [ISAP, IPdu] :=
  [IPdu?ipdu:IPDU[not (i:ICOMReq(sp))]; ICOMconf; exit]
  [not (i:ICOMReq(sp))] => Wait [ISAP, IPdu]
exit (* Connectrequest *)
endproc (* Connectrequest *)

process 

**5. Experiences and evaluation**

The specifications have been checked by tools and by thorough review. This of course doesn't exclude the possibility of errors. The specifications appear to be fairly "correct" as far as syntax and the specified behaviour are concerned. But since the term "correct" has many meanings in the context of semantics the author is aware of the fact that there may still be problems with the specifications and is happy about any comment. In particular, it wasn't possible to formally verify the protocol specifications against the service specifications, also due to the fact that it is not really clear what verification means in this context. What kinds of equivalence relation should hold between service and protocol?

The LOTOS specifications have been syntactically checked with the Hippo tool [vE88]. The semantics have been checked by performing a limited number of simulation experiments on the specifications with the same tool.

The syntax of the Estelle specifications has been checked with the Estelle-C compiler [CHA87] and also some experiments have been performed on the specifications by simulation.

The SDL specifications have been check by thorough review. Many comments have been received from readers of [HOG89] after the first publication of the specification of Inters. Some of the comments lead to corrections in the specification.

It has been experienced during the specification process that the differences between the three languages are not very big. The SDL and Estelle specifications could almost be translated one to one into another. Differences are mainly due to the different input port semantics of the two languages. SDL only has one input port per process and discards unexpected signals, while in Estelle any number if input ports per process are possible and unexpected messages may lead to deadlock.
The LOTOS specification of the Inres protocol has been produced according to the state oriented approach [VSS88]. This makes it very similar to the SDL and Estelle specifications of the Inres protocol. Many of the state names in SDL and Estelle appear as process names in the LOTOS specification. The Inres service specification on the other hand is constraint oriented. This makes it fundamentally different to the SDL and Estelle specifications of the Inres service.

6. References


