**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 [Z100], 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 FDTs (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 [SPI89], stream functions [BRO87], temporal logic [GOT91], to specify the same protocol with their approach.

In the following, two services and one protocol are described:

· the Medium service, which can be used for unreliable transmission of data units
· the Inres protocol (initiator-responder), which - with the aid of the Medium service - renders a connection-oriented service to its users
· the Inres service, which is the service rendered by the Inres protocol and the Medium service.

The services and protocols described here cannot be related to any specific layers of the OSI-BRM, although they contain some basic OSI elements. Fig. 1.1 shows the basic structure of the example.

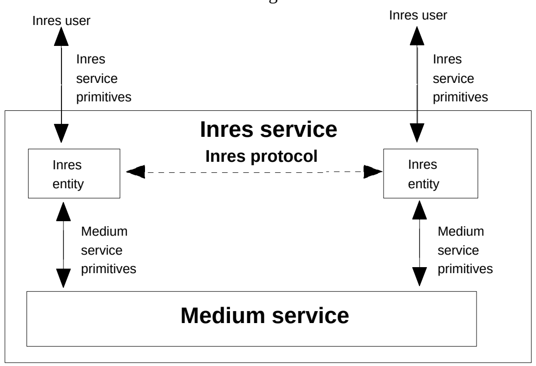



Figure 1.1 Basic architecture of the Inres system



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.


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(ISDU): data from the Initiator-user to the provider, this SP has a parameter of type ISDU
· IDATind(ISDU): data from the Provider to the Responder-user, this SP has a parameter of type ISDU
· 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]).

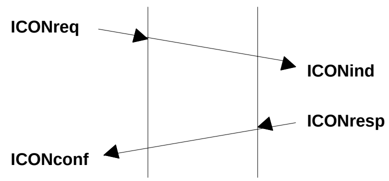



Figure 1.3a Successful connection establishment

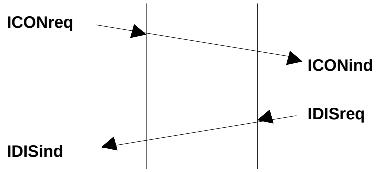



Figure 1.3b Unsuccessful connection establishment (rejection by the Responder)

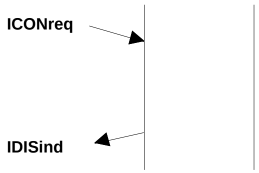



Figure 1.3c Unsuccessful connection establishment (erroneous transmission of the connection request)

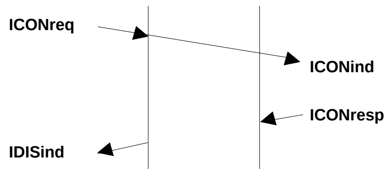



Figure 1.3d Unsuccessful connection establishment (erroneous transmission of the connection response)

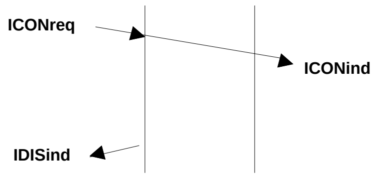



Figure 1.3e Unsuccessful connection establishment (Responder ignores connection request)

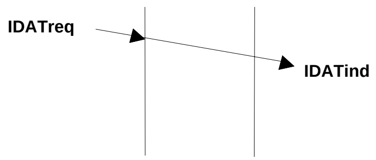



Figure 1.3f Successful data transfer

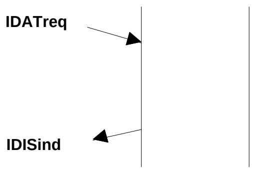





Figure 1.3g Unsuccessful data transfer (erroneous transmission of data)

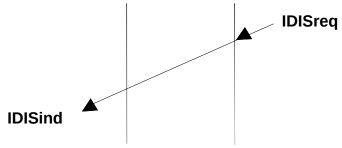



Figure 1.3h Successful disconnection

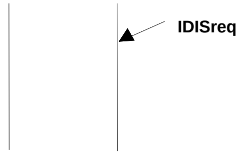



Figure 1.3i Unsuccessful disconnection (erroneous transmission of disconnection request)

### 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.

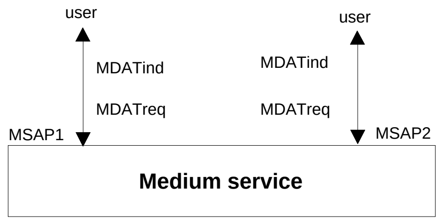



Figure 1.4 The Medium service

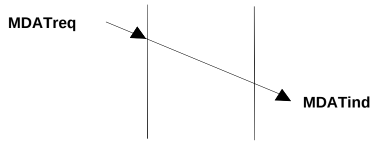



Figure 1.5a Successful data transfer

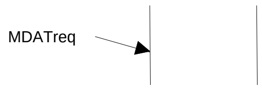



Figure 1.5b Unsuccessful data transfer (erroneous transmission of data)

### 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.



| PDU | Meaning | parameter | respective SPs |
|---|---|---|---|
| CR | connection establishment | none | ICONreq,ICONind |
| CC | connection confirmation | none | ICONresp,ICONconf |
| DT | data transfer | sequence number,ISDU | IDATreq,IDATind |
| AK | acknowledgement | sequence number | - |
| DR | disconnection | none | IDISreq,IDISind |

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.

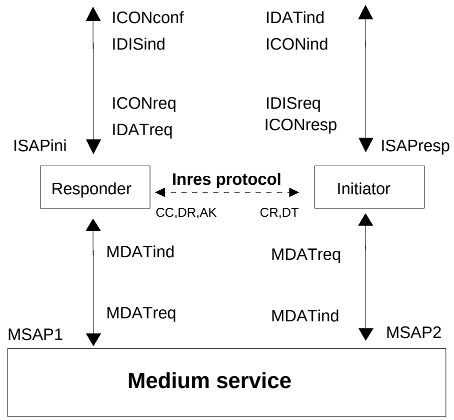



Figure 1.6 The Inres protocol

**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 diconnection 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 diconnection 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 [BHT88]

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-BRM. 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.

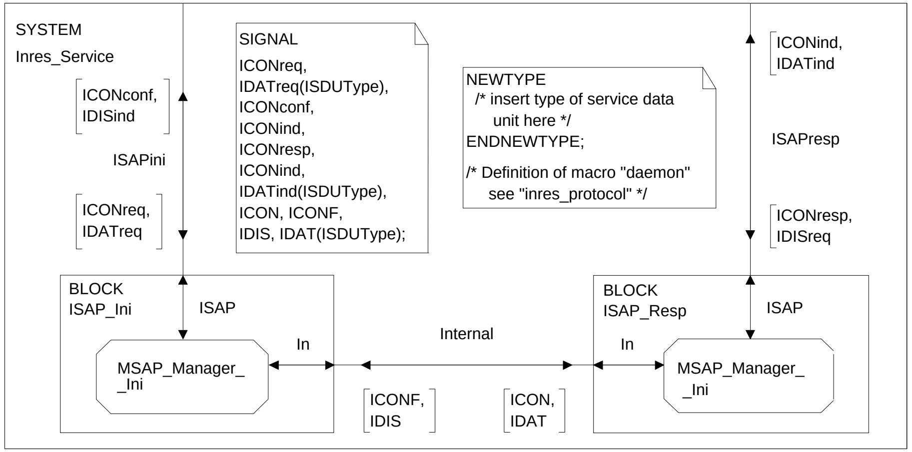
· Second, the very important aspect that the service can be seen as an abstraction of the protocol and the next lower service. This is expressed in SDL by an abstract "overview" block specification in terms of interacting processes.

### 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.



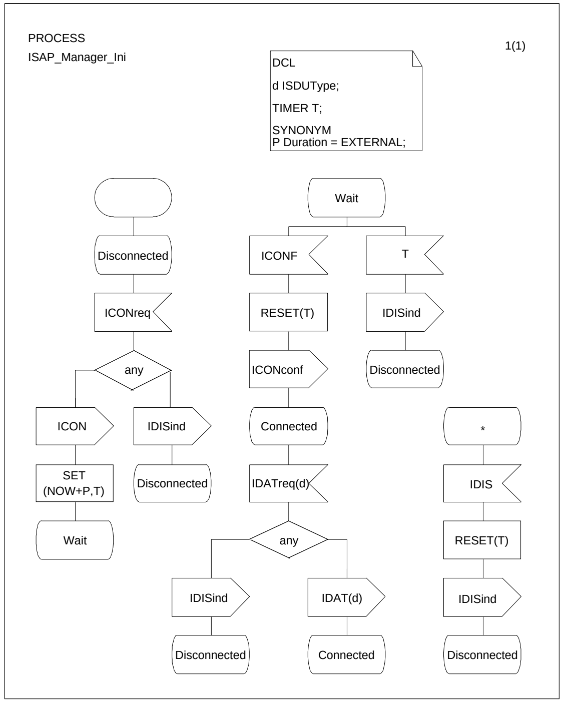
Example 2.1:




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.

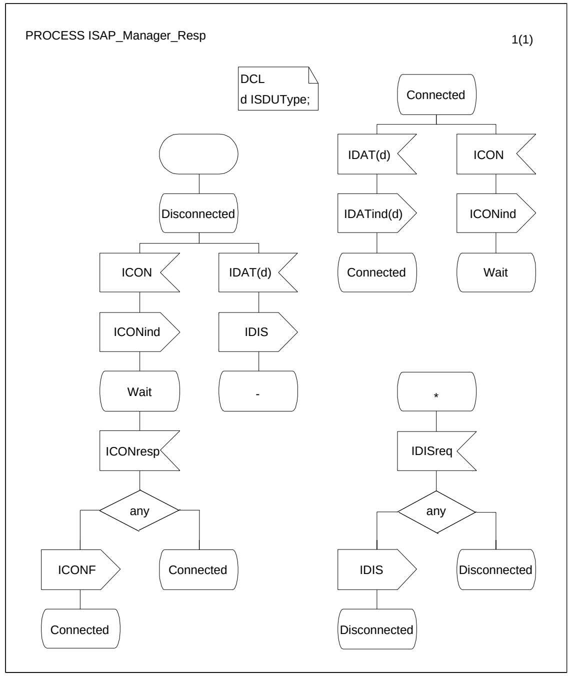
Example 2.2 shows the behaviour of the Initiator-SAP called ISAP_Manager_Ini, and Example 2.3 shows the behaviour of the Responder-SAP called ISAP_Manager_Res. ISAP_Manager_Ini and ISAP_Manager_Res can communicate through a channel to establish the global behaviour of the service.

Example 2.2:




Example 2.3:






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_Ini to ISAP_Manager_Res. 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 ICONind - 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 ICONresp 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 ICONreq, ISAP_Manager_Res receives an ICON, issues an ICONind 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 attempt 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
    ICONreq,
    IDATreq( ISDUType),
    ICONconf,
    ICONind,
    ICONresp,
    IDISind,
    IDISreq,
    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 Intern
  from ISAP_Ini to ISAP_Resp
    with ICON, IDAT;
  from ISAP_Resp to ISAP_Ini
    with ICONF, IDIS;
endchannel Intern;

channel ISAPini
  from ISAP_Ini to env
    with ICONconf, IDISind;
  from env to ISAP_Ini
    with ICONreq, IDATreq;
endchannel ISAPini;
```



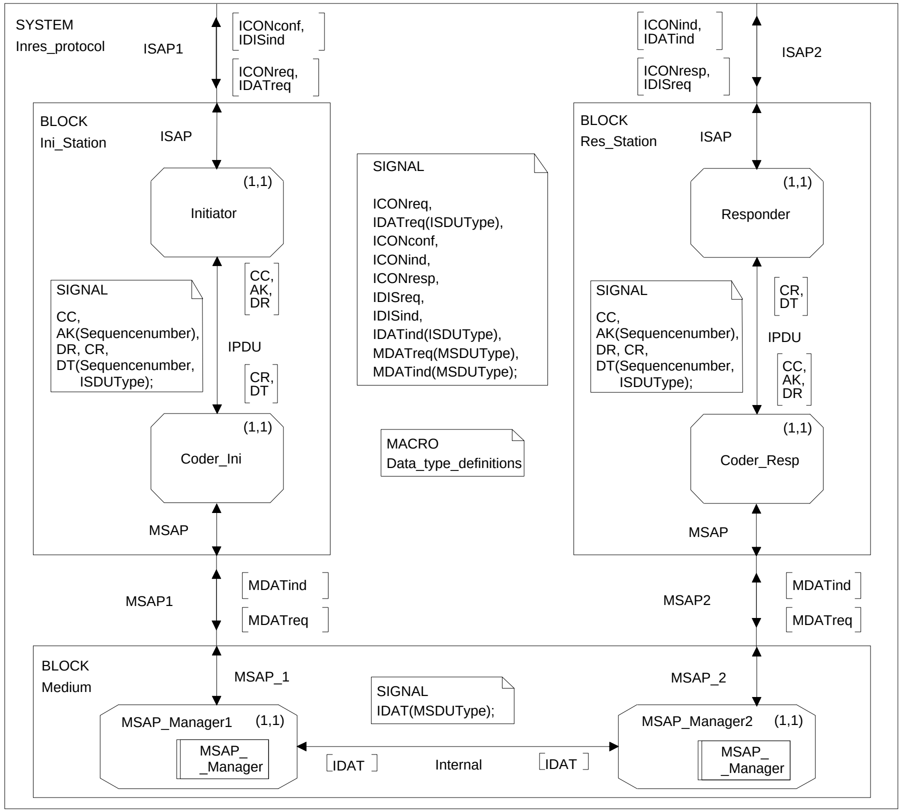
```
block ISAP_Resp referenced;


block ISAP_Ini referenced;
endsystem Inres_Service;


block ISAP_Resp;

  connect Intern and Internal;

  connect ISAPresp and ISAP;

  signalroute Internal
    from ISAP_Manager_Resp to env
      with ICONF, IDIS;
    from env to ISAP_Manager_Resp
      with ICON, IDAT;

  signalroute ISAP
    from ISAP_Manager_Resp to env
      with ICONind, IDATind;
    from env to ISAP_Manager_Resp
      with ICONresp, IDISreq;


process ISAP_Manager_Resp referenced;
endblock ISAP_Resp;



block ISAP_Ini;

  connect Intern and Internal;

  connect ISAPini and ISAP;

  signalroute ISAP
    from ISAP_Manager_Ini to env
      with ICONconf, IDISind;
    from env to ISAP_Manager_Ini
      with ICONreq, IDATreq;

  signalroute Internal
    from ISAP_Manager_Ini to env
      with ICON, IDAT;
    from env to ISAP_Manager_Ini
      with ICONF, IDIS;


process ISAP_Manager_Ini referenced;
endblock ISAP_Ini;



process ISAP_Manager_Resp;

  dcl
    d ISDUType;

  start;
      nextstate Disconnected;

  state Wait;
    input ICONresp;
      decision ANY;
      ( EITHER) :
        output ICONF;
        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);
```

```
      output IDATind( d);
      nextstate Connected;
    input ICON;
      output ICONind;
      nextstate Wait;

  state * ;
    input IDISreq;
      decision ANY;
      ( EITHER) :
        output IDIS;
        nextstate Disconnected;
      ( OR) :
        nextstate Disconnected;
      enddecision;
endprocess ISAP_Manager_Resp;



process ISAP_Manager_Ini;

  dcl
    d ISDUType;

  timer
    T;

  synonym P Duration = external;

  start;
      nextstate Disconnected;

  state Disconnected;
    input ICONreq;
      decision ANY;
      ( EITHER) :
        output ICON;
        set( now + P, T);
        nextstate Wait;
      ( OR) :
        output IDISind;
        nextstate Disconnected;
      enddecision;

  state Connected;
    input IDATreq( d);
      decision ANY;
      ( EITHER) :
        output IDISind;
        nextstate Disconnected;
      ( OR) :
        output IDAT( d);
        nextstate Connected;
      enddecision;

  state Wait;
    input ICONF;
      reset( T);
      output ICONconf;
      nextstate Connected;
    input T;
      output IDISind;
      nextstate Disconnected;

  state * ;
    input IDIS;
      reset( T);
      output IDISind;
      nextstate Disconnected;
endprocess ISAP_Manager_Ini;
```

### 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)






Example 2.4:

MACRODEFINITION Datatypedefinitions

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

**NEWTYPE** ISDUType
/* Here the data type of the service data unit is specified */
**ENDNEWTYPE** ISDUType;

**NEWTYPE** IPDUType
**LITERALS** CR, CC, DR, DT, AK;
**ENDNEWTYPE** IPDUType;

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

data ISDUType;
**ENDNEWTYPE** MSDUType;

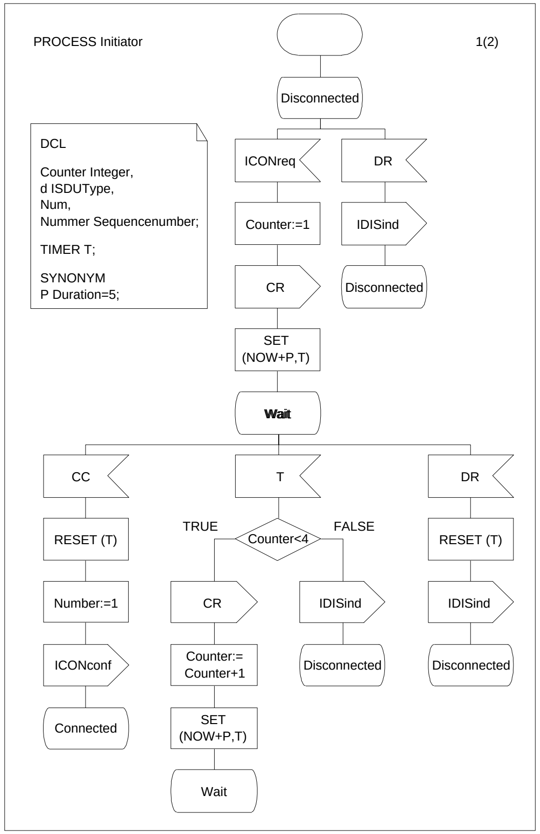
**ENDMACRO** Datatypedefinitions;

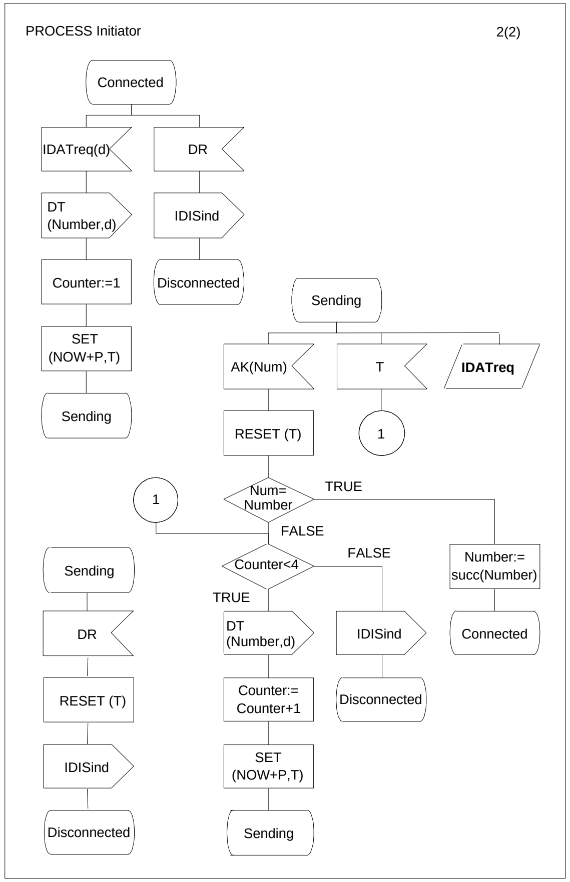
The specification consists of three basic parts, all three of which are modelled by blocks: the two protocol entities Station_Ini and Station_Res, and the service provider Medium.

Each Station consists of two processes. The Coder 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 Coder_Ini and Coder_Res.





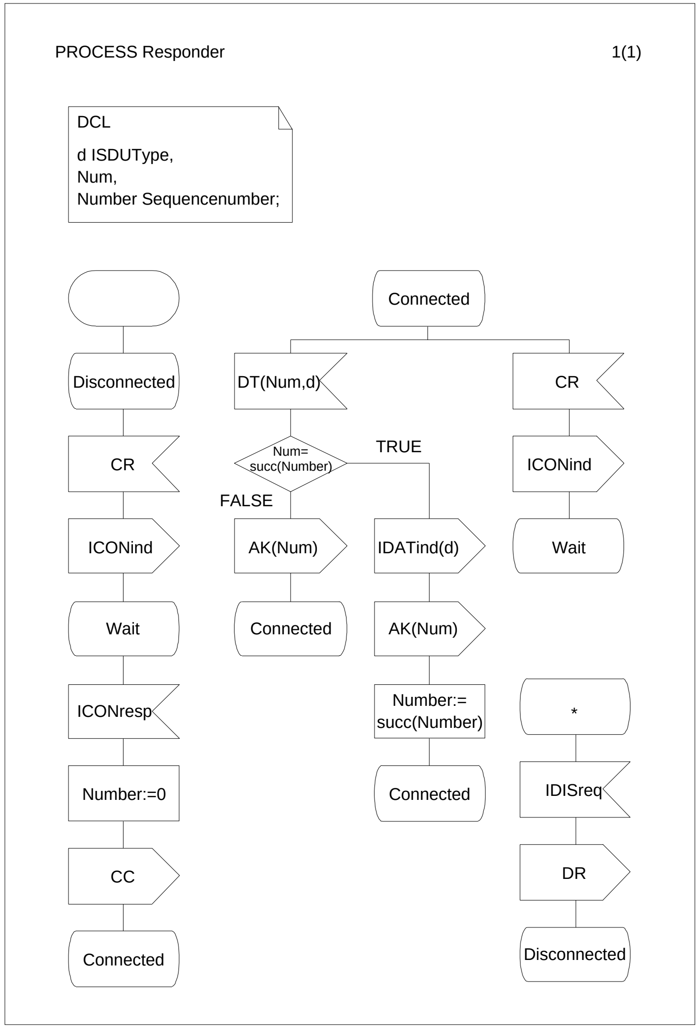



Example 2.5:

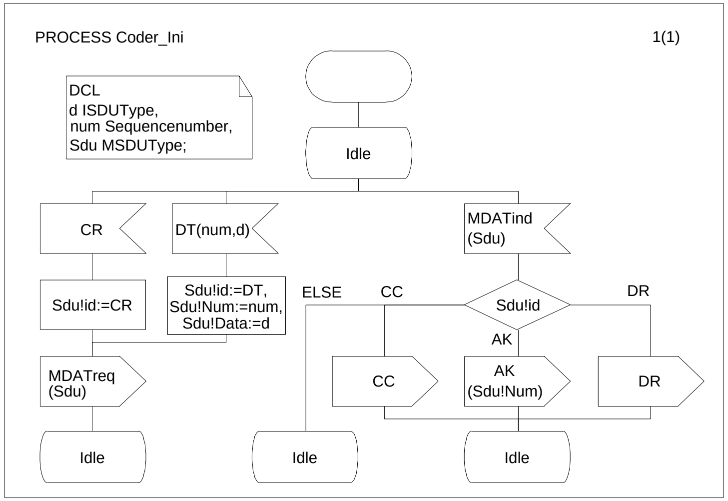
In the most general case the Coder processes may have additional duties. According to [ISO 7498] (more precisely Section 5.7.4 in [ISO 7498]) these processes may handle the connection setup and maintenance of the next lower layer. More on this topic is given in [BHS91].

The SDL specification of the Inres 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.





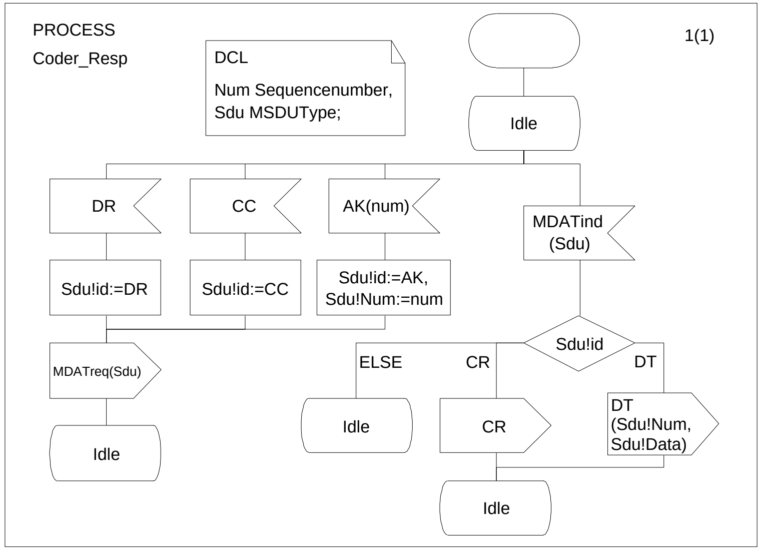

Example 2.6:



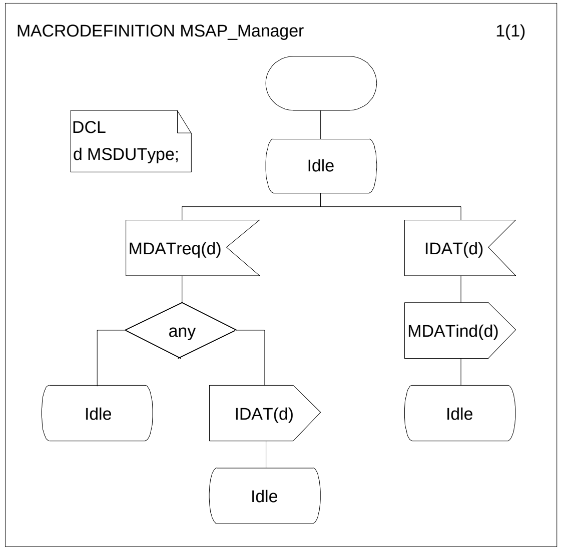

Example 2.7:



Example 2.8:




Example 2.9:




### 2.4 The Inres protocol and Medium service in SDL/PR

```
system INRES;
  signal
```

```
    ICONreq,
    IDATreq( ISDUTyp),
    ICONconf,
    ICONind,
    ICONresp,
    IDISreq,
    IDISind,
    IDATind( ISDUTyp),
    MDATreq( MSDUTyp),
    MDATind( MSDUTyp);

channel MSAP1
  from Ini_Station to Medium
    with MDATreq;
  from Medium to Ini_Station
    with MDATind;
endchannel MSAP1;

channel ISAP1
  from Ini_Station to env
    with ICONconf, IDISind;
  from env to Ini_Station
    with ICONreq, IDATreq;
endchannel ISAP1;

channel MSAP2
  from Res_Station to Medium
    with MDATreq;
  from Medium to Res_Station
    with MDATind;
endchannel MSAP2;

channel ISAP1
  from Res_Station to env
    with ICONind, IDATind;
  from env to Res_Station
    with ICONresp, IDISreq;
endchannel ISAP1;



block Ini_Station referenced;



block Medium referenced;



block Res_Station referenced;

  newtype Sequencenumber
    literals 0, 1
    operators
      succ :
        Sequencenumber -> Sequencenumber;
    axioms
      succ( 0) == 1;
      succ( 1) == 0;
  endnewtype Sequencenumber;

  newtype ISDUType
/* Here the data type of the service data unit is specified */
  endnewtype ISDUType;

  newtype IPDUType
    literals CR, CC, DR, DT, AK
  endnewtype IPDUType;

  newtype MSDUType
    struct
      id IPDUType;
      Num Sequencenumber;
      Daten ISDUType;
  endnewtype MSDUType;
endsystem INRES;



block Ini_Station;

  signal
    CC,
    AK( Sequencenumber),
    DR,
    CR,
    DT( Sequencenumber, ISDUType);

  connect ISAP1 and ISAP;

  connect MSAP1 and MSAP;
```



```
  signalroute MSAP
    from Coder_Ini to env
      with MDATreq;
    from env to Coder_Ini
      with MDATind;

  signalroute IPDU
    from Initiator to Coder_Ini
      with CR, DT;
    from Coder_Ini to Initiator
      with CC, AK, DR;

  signalroute ISAP
    from Initiator to env
      with ICONconf, IDISind;
    from env to Initiator
      with ICONreq, IDATreq;



process Coder_Ini (1, 1) referenced;



process Initiator (1, 1) referenced;
endblock Ini_Station;




block Medium;

  signal
    IDAT( MSDUType);

  connect MSAP1 and MSAP_1;

  connect MSAP2 and MSAP_2;

  signalroute MSAP_1
    from MSAP_Manager1 to env
      with MDATind;
    from env to MSAP_Manager1
      with MDATreq;

  signalroute MSAP_2
    from MSAP_Manager2 to env
      with MDATind;
    from env to MSAP_Manager2
      with MDATreq;

  signalroute Internal
    from MSAP_Manager1 to MSAP_Manager2
      with IDAT;
    from MSAP_Manager2 to MSAP_Manager1
      with IDAT;



process MSAP_Manager2 (1, 1) referenced;



process MSAP_Manager1 (1, 1) referenced;
endblock Medium;




block Res_Station;

  signal
    CC,
    AK( Sequencenumber),
    DR,
    CR,
    DT( Sequencenumber, ISDUType);

  connect ISAP2 and ISAP;

  connect MSAP2 and MSAP;

  signalroute MSAP
    from Coder_Resp to env
      with MDATreq;
    from env to Coder_Resp
      with MDATind;

  signalroute IPDU
    from Responder to Coder_Resp
```

```
      with CC, AK, DR;
    from Coder_Resp to Responder
      with CR, DT;

  signalroute 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_Ini;

  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_Ini;




process Initiator;

  dcl
    Counter Integer,
    d ISDUType,
    Num,
    Nummer 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;
      output IDISind;
      nextstate Disconnected;

  state Wait;
    input CC;
```



```
      reset( T);
      task Number := 1;
      output ICONconf;
      nextstate Connected;
    input T;
      decision Counter < 4;
      ( TRUE) :
        output CR;
        task Counter := Counter + 1;
        set( now + P, T);
        nextstate Wait;
      ( FALSE) :
        output IDISind;
        nextstate Disconnected;
      enddecision;
    input DR;
      reset( T);
      output IDISind;
      nextstate Disconnected;

  state Connected;
    input IDATreq( d);
      output DT( Number, d);
      task Counter := 1;
      set( now + P, T);
      nextstate Sending;
    input DR;
      output IDISind;
      nextstate Disconnected;

  state Sending;
    input T;
      grs0 :
      decision Counter < 4;
      ( TRUE) :
        output DT( Number, d);
        task Zaehler := Zaehler + 1;
        set( now + P, T);
        nextstate Sending;
      ( FALSE) :
        output IDISind;
        nextstate Disconnected;
      enddecision;
    input AK( Num);
      reset( T);
      decision Num = Number;
      ( FALSE) :
        join grs0;
      ( TRUE) :
        task Number := succ( Number);
        nextstate Connected;
      enddecision;
    input DR;
      reset( T);
      output IDISind;
      nextstate Disconnected;
    save IDATreq;
endprocess Initiator;




process MSAP_Manager2;

  dcl
    d MSDUTyp;

  start;
      nextstate Idle;

  state Idle;
    input MDATreq( d);
      decision ANY;
      ( EITHER) :
        nextstate Idle;
      ( OR) :
        output IDAT( d);
        nextstate Idle;
      enddecision;
    input IDAT( d);
      output MDATind( d);
      nextstate Idle;
endprocess MSAP_Manager2;




process MSAP_Manager1;

  dcl
    d MSDUType;
```

```
  start;
      nextstate Idle;

  state Idle;
    input MDATreq( d);
      decision ANY;
      ( EITHER) :
        nextstate Idle;
      ( OR) :
        output IDAT( d);
        nextstate Idle;
      enddecision;
    input IDAT( d);
      output MDATind( d);
      nextstate Idle;
endprocess MSAP_Manager1;



process Coder_Resp;

  dcl
    Num Sequencenumber,
    Sdu MSDUType;

  start;
      nextstate Idle;

  state Idle;
    input DR;
      task Sdu!id := DR;
      grs0 :
      output MDATreq( Sdu);
      nextstate Idle;
    input CC;
      task Sdu!id := CC;
      join grs0;
    input AK( Num);
      task Sdu!id := AK,
        Sdu!Num := Num;
      join grs0;
    input MDATind( Sdu);
      decision Sdu!id;
      ( CR) :
        output CR;
        grs1 :
        nextstate Idle;
      ( DT) :
        output DT( Sdu!Num, Sdu!Data);
        join grs1;
      else :
        nextstate Bereit;
      enddecision;
endprocess Coder_Resp;



process Responder;

  dcl
    d ISDUType,
    Num,
    Number Sequencenumber;

  start;
      nextstate Disconnected;

  state Disconnected;
    input CR;
      output ICONind;
      nextstate Wait;

  state Wait;
    input ICONresp;
      task Number := 0;
      output CC;
      nextstate Connected;

  state Connected;
    input DT( Num, d);
      decision Num = succ( Number);
      ( FALSE) :
        output AK( Num);
        nextstate Connected;
      ( TRUE) :
        output IDATind( d);
        output AK( Num);
        task Number := succ( Number);
```



```
        nextstate Connected;
      enddecision;
    input CR;
      output ICONind;
      nextstate Wait;
endprocess Responder;
```

## 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 itsself 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.

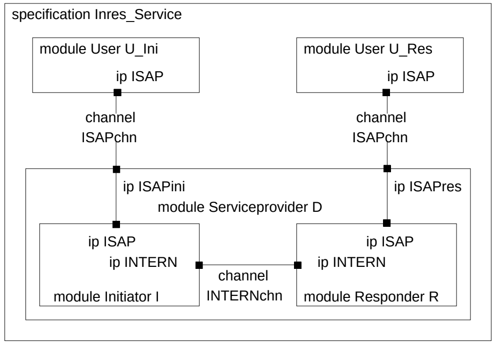



Figure 3.1

```
specification Inres_service;
default individual queue;
timescale seconds;
type ISDUType = integer; {Pascal type definitions}
channel ISAPchn(User,Service);
     by User :
          ICONreq;
          ICONresp;
          IDATreq(ISDU : ISDUType);
          IDISreq;
     by Service :
          ICONconf;
          ICONind;
          IDATind(ISDU : ISDUType);
          IDISind;
module User systemprocess;
     ip ISAP : ISAPchn(User);                                             end;
body User_Body for User;                                                  end;
module Service_Provider systemprocess;
     ip ISAPini : ISAPchn(Service);
        ISAPres : ISAPchn(Service);                                       end;
body Service_Provider_Body for Service_Provider;
     channel INTERNchn(Ini,Res);
          by Ini :
               ICON;
               IDAT(ISDU : ISDUType);
          by Res :
               ICONF;
               IDIS;
     module Initiator process;
          ip USER : ISAPchn(Service);
             INTERN : INTERNchn(Ini);                                     end;
     body Initiator_Body for Initiator;
          state DISCONNECTED, WAIT, CONNECTED;
          stateset
               anystate = [DISCONNECTED, WAIT, CONNECTED];
               ignoreICONreq = [WAIT, CONNECTED];
               ignoreIDATreq = [DISCONNECTED, WAIT];
               ignoreICONF = [DISCONNECTED, CONNECTED];
          initialize to DISCONNECTED                                begin end;
          trans
               from DISCONNECTED to WAIT {1}
                    when USER.ICONreq                                    begin
                         output INTERN.ICON                              end;
               from DISCONNECTED to same {2}
                    when USER.ICONreq                                    begin
                         output USER.IDISind                             end;
               from WAIT to DISCONNECTED {3}
                    delay (5)                                            begin
                         output USER.IDISind                             end;
               from WAIT to CONNECTED {4}
                    when INTERN.ICONF                                    begin
                         output USER.ICONconf                            end;
               from CONNECTED to same {5}
                    when USER.IDATreq(ISDU)                              begin
                         output INTERN.IDAT(ISDU)                        end;
               from CONNECTED to DISCONNECTED {6}
                    when USER.IDATreq(ISDU)                              begin
                         output USER.IDISind                             end;
               from anystate to DISCONNECTED {7}
                    when INTERN.IDIS                                     begin
                         output USER.IDISind                             end;

               from ignoreICONreq to same {8}
```



```
                    when USER.ICONreq                              begin   end;
               from ignoreIDATreq to same {9}
                    when USER.IDATreq                              begin   end;
               from ignoreICONF to same {10}
                    when INTERN.ICONF                      begin   end;   end;
     module Responder process;
          ip USER : ISAPchn(Service);
             INTERN : INTERNchn(Res);                                     end;
     body Responder_Body for Responder;
          state DISCONNECTED, WAIT, CONNECTED;
          stateset
               anystate = [DISCONNECTED, WAIT, CONNECTED];
               ignoreICONresp = [DISCONNECTED, CONNECTED];
               ignoreICON = [WAIT, CONNECTED];
               ignoreIDAT = [WAIT];
          initialize to DISCONNECTED                               begin  end;
          trans
               from DISCONNECTED to WAIT {11}
                    when INTERN.ICON                                     begin
                         output USER.ICONind                              end;
               from WAIT to CONNECTED {12}
                    when USER.ICONresp                                   begin
                         output INTERN.ICONF                              end;
               from WAIT to CONNECTED {13}
                    when USER.ICONresp                             begin   end;
               from CONNECTED to same {14}
                    when INTERN.IDAT(ISDU)                               begin
                         output USER.IDATind(ISDU)                        end;
               from DISCONNECTED to same {15}
                    when INTERN.IDAT(ISDU)                               begin
                         output INTERN.IDIS                               end;
               from CONNECTED to WAIT {16}
                    when INTERN.ICON                                     begin
                         output USER.ICONind                              end;
               from anystate to DISCONNECTED {17}
                    when USER.IDISreq                                    begin
                         output INTERN.IDIS                               end;
               from anystate to DISCONNECTED {18}
                    when USER.IDISreq                              begin   end;
               from ignoreICONresp to same {19}
                    when USER.ICONresp                             begin   end;
               from ignoreICON to same {20}
                    when INTERN.ICON                               begin   end;
               from ignoreIDAT to same {21}
                    when INTERN.IDAT                       begin   end;   end;
     modvar
          I : Initiator;
          R : Responder;
     initialize                                                          begin
          init I with Initiator_Body;
          init R with Responder_Body;
          attach ISAPini to I.USER;
          attach ISAPres to R.USER;
          connect I.INTERN to R.INTERN;                            end;   end;
modvar
     U_Ini,U_Res : User;
     SP : Service_Provider;
initialize                                                               begin
          init U_Ini with User_Body;
          init U_Res with User_Body;
          init SP with Service_Provider_Body
          connect U_Ini.ISAP to SP.ISAPini;
          connect U_Res.ISAP to SP.ISAPres;                        end;   end.
```

### 3.2 The Inres protocol and Medium service in Estelle

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.

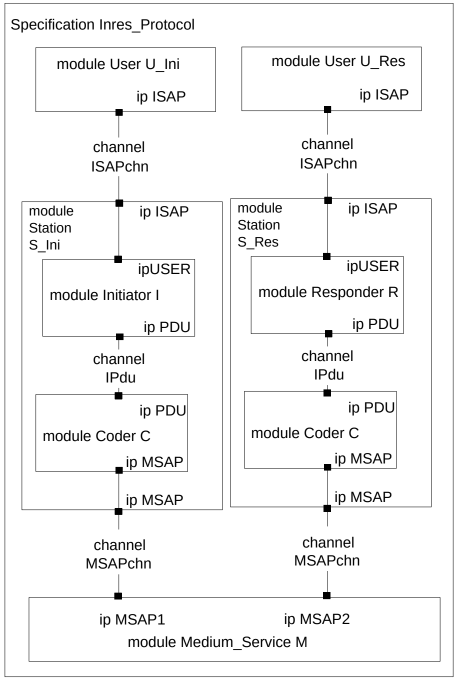



Figure 3.2

```
specification Inres_Protocol;
default individual queue;
timescale seconds;
type ISDUType = integer; {Pascal data type definition. Use integer for compilation.}
type Sequencenumber = 0..1;
type PduType = (CR,CC,DT,AK,DR);
type MSDUType = record id : PduType;
                     num : Sequencenumber;
                     data : ISDUType;                                end;
channel ISAPchn(User,Station);
     by User :
          ICONreq;
          ICONresp;
          IDATreq(ISDU : ISDUType);
          IDISreq;
     by Station :
          ICONconf;
          ICONind;
          IDATind(ISDU : ISDUType);
          IDISind;
channel MSAPchn(Station,Medium_Service);
     by Station :
          MDATreq(MSDU : MSDUType);
     by Medium_Service :
          MDATind(MSDU : MSDUType);
module User systemprocess;
     ip ISAP : ISAPchn(User);                                        end;
body User_Body1 for User;                                            end;
body User_Body2 for User;                                            end;
module Medium_Service systemprocess;
     ip MSAP1 : MSAPchn(Medium_Service);
        MSAP2 : MSAPchn(Medium_Service);                             end;
body Medium_Body for Medium_Service;
```



```
     trans
          when MSAP1.MDATreq(MSDU)                                   begin
               output MSAP2.MDATind(MSDU)                            end;
          when MSAP2.MDATreq(MSDU)                                   begin
               output MSAP1.MDATind(MSDU)                            end;
          when MSAP1.MDATreq(MSDU)                            begin  end;
          when MSAP2.MDATreq(MSDU)                     begin  end;   end;
module Station systemprocess;
     ip ISAP : ISAPchn(Station);
        MSAP : MSAPchn(Station);                                     end;
body Station_Ini_Body for Station;
     channel IPdu(Initiator,Coder);
          by Initiator :
               CR;
               DT(Num:Sequencenumber;ISDU:ISDUType);
          by Coder :
               CC;
               AK(Num:Sequencenumber);
               DR;
     module Initiator process;
          ip USER : ISAPchn(Station);
             PDU : IPdu(Initiator);                                  end;
     body Initiator_Body for Initiator;
          var olddata : ISDUType;
              counter : 0..4;
              number : Sequencenumber;
          function succ(Number:Sequencenumber) : Sequencenumber;            begin
               if Number = 0 then succ :=1
               else succ := 0                                               end;
          state DISCONNECTED,WAIT,CONNECTED,SENDING;
          stateset
               anystate = [DISCONNECTED,WAIT,CONNECTED,SENDING];
               ignoreICONreq = [WAIT,CONNECTED,SENDING];
               ignoreIDATreq = [DISCONNECTED,WAIT];
               ignoreCC = [DISCONNECTED,CONNECTED,SENDING];
               ignoreAK = [DISCONNECTED,WAIT,CONNECTED];
          initialize to DISCONNECTED                          begin  end;
          trans
               from DISCONNECTED to WAIT
                    when USER.ICONreq                                begin
                         counter := 0;
                         output PDU.CR                               end;
               from WAIT to CONNECTED
                    when PDU.CC                                      begin
                         number := 1;
                         counter := 0;
                         output USER.ICONconf                        end;
               from WAIT
                    delay(5)
                         provided counter < 4
                              to same                                begin
                                   output PDU.CR;
                                   counter := counter + 1            end;
                         provided otherwise
                              to DISCONNECTED                        begin
                                   output USER.IDISind               end;
               from CONNECTED to SENDING
                    when USER.IDATreq(ISDU)                          begin
                         output PDU.DT(number,ISDU);
                         olddata := ISDU                             end;
               from SENDING
                    when PDU.AK(Num)
                         provided Num = number
                              to CONNECTED                           begin
                                   number := succ(number)            end;
                         provided (Num <> number)
                                   and (counter < 4)
                              to same                                begin
                                   output PDU.DT(number,olddata);
                                   counter := counter + 1            end;
                         provided otherwise
                              to DISCONNECTED                        begin
                                   output USER.IDISind               end;
               from SENDING
                    delay(5)
                         provided counter < 4
                              to same                                begin
                                   output PDU.DT(number,olddata);
                                   counter := counter + 1            end;
                         provided otherwise
                              to DISCONNECTED                        begin
                                   output USER.IDISind               end;
               from anystate to DISCONNECTED
                    when PDU.DR                                      begin
                         output USER.IDISind                         end;
               from anystate to same
                    when USER.ICONresp                        begin   end;
                    when USER.IDISreq                         begin  end;
               from ignoreICONreq to same
                    when USER.ICONreq                         begin  end;
               from ignoreIDATreq to same
```

```
                    when USER.IDATreq                            begin  end;
                from ignoreCC to same
                    when PDU.CC                                  begin  end;
                from ignoreAK to same
                    when PDU.AK                          begin   end;   end;
        module Coder process;
            ip PDU : IPdu(Coder);
               MSAP : MSAPchn(Station);                                 end;
        body Coder_Body for Coder;
            var MSDU : MSDUType;
            trans
                when PDU.CR                                             begin
                    MSDU.id := CR;
                    output MSAP.MDATreq(MSDU)                           end;
                when PDU.DT(Num,ISDU)                                   begin
                    MSDU.id := DT;
                    MSDU.num := Num;
                    MSDU.data := ISDU;
                    output MSAP.MDATreq(MSDU)                           end;
                when MSAP.MDATind(MSDU)                                 begin
                    case MSDU.id of
                    CC: output PDU.CC;
                    AK: output PDU.AK(MSDU.num);
                    DR: output PDU.DR;                   end;   end;    end;
        modvar
            I : Initiator;
            C : Coder;
        initialize                                                      begin
            init I with Initiator_Body;
            init C with Coder_Body;
            attach ISAP to I.USER;
            attach MSAP to C.MSAP;
            connect I.PDU to C.PDU;                             end;    end;
    body Station_Res_Body for Station;
        channel IPdu(Responder,Coder);
            by Responder :
                CC;
                AK(Num:Sequencenumber);
                DR;
            by Coder :
                CR;
                DT(Num:Sequencenumber;ISDU:ISDUType);
        module Responder process;
            ip USER : ISAPchn(Station);
               PDU : IPdu(Responder);                                   end;
        body Responder_Body for Responder;
            state DISCONNECTED,WAIT,CONNECTED;
            var number : Sequencenumber;
            function succ(Number:Sequencenumber) : Sequencenumber;             begin
                if Number = 0 then succ :=1
                else succ := 0                                                 end;
            stateset
                anystate = [DISCONNECTED,WAIT,CONNECTED];
                ignoreICONresp = [DISCONNECTED,CONNECTED];
                ignoreCR = [WAIT];
                ignoreDT = [DISCONNECTED,WAIT];
            initialize to DISCONNECTED                                  begin   end;
            trans
                from DISCONNECTED to WAIT
                    when PDU.CR                                         begin
                        output USER.ICONind                             end;
                from WAIT to CONNECTED
                    when USER.ICONresp                                  begin
                        number := 0;
                        output PDU.CC                                   end;
                from CONNECTED to same
                    when PDU.DT(Num, ISDU)
                      provided Num = succ(number)                       begin
                        output USER.IDATind(ISDU);
                        output PDU.AK(Num);
                        number := succ(number)                          end;
                      provided Num = number                             begin
                        output PDU.AK(Num)                              end;
                from CONNECTED to WAIT
                    when PDU.CR                                         begin
                        output USER.ICONind                             end;
                from anystate to DISCONNECTED
                    when USER.IDISreq                                   begin
                        output PDU.DR                                   end;
                from anystate to same
                    when USER.ICONreq                           begin   end;
                    when USER.IDATreq                           begin   end;
                from ignoreICONresp to same
                    when USER.ICONresp                          begin   end;
                from ignoreCR to same
                    when PDU.CR                                 begin   end;
                from ignoreDT to same
                    when PDU.DT                          begin   end;   end;
        module Coder process;
            ip PDU : IPdu(Coder);
               MSAP : MSAPchn(Station);                                 end;
```



```
        body Coder_Body_Res for Coder;
            var MSDU : MSDUType;
            trans
                when PDU.CC                                             begin
                    MSDU.id := CR;
                    output MSAP.MDATreq(MSDU)                           end;
                when PDU.AK(Num)                                        begin
                    MSDU.id := AK;
                    MSDU.num := Num;
                    output MSAP.MDATreq(MSDU)                           end;
                when PDU.DR                                             begin
                    MSDU.id := DR;
                    output MSAP.MDATreq(MSDU)                           end;
                when MSAP.MDATind(MSDU)                                 begin
                    case MSDU.id of
                    CR: output PDU.CR;
                    DT: output PDU.DT(MSDU.num,MSDU.data);      end;   end;   end;
        modvar
            R : Responder;
            C : Coder;
        initialize                                                      begin
            init R with Responder_Body;
            init C with Coder_Body_Res;
            attach ISAP to R.USER;
            attach MSAP to C.MSAP;
            connect R.PDU to C.PDU;                             end;    end;
modvar
    U_Ini,U_Res : User;
    S_Ini,S_Res : Station;
    M : Medium_Service;
initialize                                                              begin
            init U_Ini with User_Body1;
            init U_Res with User_Body2;
            init S_Ini with Station_Ini_Body;
            init S_Res with Station_Res_Body;
            init M with Medium_Body;
            connect U_Ini.ISAP to S_Ini.ISAP;
            connect U_Res.ISAP to S_Res.ISAP;
            connect M.MSAP1 to S_Ini.MSAP;
            connect M.MSAP2 to S_Res.MSAP;                      end;    end.
```

## 4. Formal specification of Inres in LOTOS

### 4.1 The Inres service in LOTOS

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.

```
specification Inres_service[ISAPini,ISAPres]:noexit
type ISDUType is

(* library Boolean type is not necessary *)

sorts ISDU
opns  data1,data2,data3,data4,data5: -> ISDU
endtype (* ISDUType *)

type InresSpType is ISDUType
sorts SP
opns ICONreq,ICONind,
      ICONresp,ICONconf,
      IDISreq,IDISind :       -> SP
      IDATreq,IDATind : ISDU  -> SP
endtype (* InresSpType *)

behaviour
  ( ICEPini[ISAPini]
    |||
    ICEPres[ISAPres]
  )
  ||
  EndtoEnd[ISAPini,ISAPres]
```

```
where

  process ICEPini[g] :noexit:=
    ( ConnectionphaseIni[g]
      >>
      DataphaseIni[g]
    )
    [>
    DisconnectionIni[g]

  where

    process ConnectionphaseIni[g] :exit:=
      g! ICONreq;
      g! ICONconf;
      exit
    endproc (* ConnectionphaseIni *)

    process DataphaseIni[g] :noexit:=
      g! IDATreq? par:ISDU;
      DataphaseIni[g]
    endproc (* DataphaseIni *)

    process DisconnectionIni[g] :noexit:=
      g!IDISind;
      ICEPini[g]
    endproc (* DisconnectionIni *)

  endproc (* ICEPini *)


  process ICEPres[g] :noexit:=
    ( ConnectionphaseRes[g]
      >>
      DataphaseRes[g]
    )
    [>
    DisconnectionRes[g]

  where

    process ConnectionphaseRes[g] :exit:=
      g!ICONind;
      g!ICONresp;
      exit
    endproc (* ConnectionphaseRes *)

    process DataphaseRes[g] :noexit:=
      g! IDATind? par:ISDU;
      DataphaseRes[g]
    endproc (* DataphaseRes *)

    process DisconnectionRes[g] :noexit:=
      g!IDISreq;
      ICEPres[g]
    endproc (* DisconnectionRes *)

  endproc.(* ICEPres *)


  process EndtoEnd[ini,res] :noexit:=
    ( ConnectionphaseEte[ini,res]
      >>
      DataphaseEte[ini,res]
    )
    [>
    DisconnectionEte[ini,res]

  where

    process ConnectionphaseEte[ini,res] :exit:=
      ( ini! ICONreq;
        res! ICONind;
        exit
      )
      |||
      ( res! ICONresp;
        ini! ICONconf;
        exit
      )
    endproc (* ConnectionphaseEte *)

    process DataphaseEte[ini,res] :noexit:=
      ini! IDATreq? par:ISDU;
      res! IDATind! par:ISDU;
      DataphaseEte[ini,res]
    endproc (* DataphaseEte *)

    process DisconnectionEte[ini,res] :noexit:=
      res! IDISreq;
```



```
      ini! IDISind;
      EndtoEnd[ini,res]
      []
      i;                       (* termination by provider *)
      ini! IDISind;
      EndtoEnd[ini,res]
    endproc (* DisconnectionEte *)

  endproc (* EndtoEnd *)

endspec (* Inres_service *)
```

### 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.

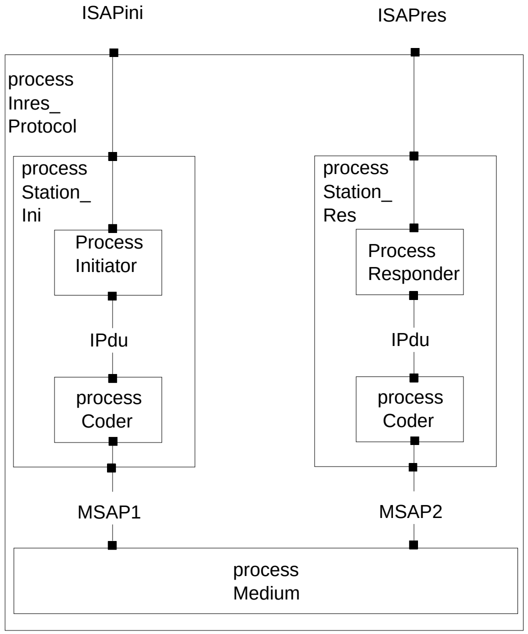



Figure 4.1 Basic architecture of the Inres protocol in LOTOS

```
specification Inres_Protocol[ISAPini,ISAPres]:noexit

library Boolean
endlib

type DecNumb is Boolean
sorts DecNumb
opns
  0                 :                      -> DecNumb
  s                 : DecNumb              -> DecNumb
  1,2,3,4,5,6,7,8,9 :                      -> DecNumb
  _==_, _<_,
  _<=_, _>=_, _>_   : DecNumb , DecNumb  -> Bool

eqns forall x,y: DecNumb
  ofsort Bool

    x == x = true;
    s(x) == s(y) = x == y;
    s(x) == 0 = false;
    0 == s(y) = false;
    x < x = false;
    s(x) < s(y) = x < y;
    0 < s(y) = true;
    s(x) < 0 = false;
```

```
    x <= y  = (x < y) or (x == y);
    x >= y  = not (x < y) ;
    x > 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(0))))))));
    9 = s(s(s(s(s(s(s(s(s(0)))))))));
endtype (* DecNumb *)

type ISDUType is
sorts ISDU
opns   data1,data2,data3,data4,data5 : -> ISDU
endtype (* ISDUType *)

type Sequencenumber is Boolean
sorts Sequencenumber
opns
  0          :                                -> Sequencenumber
  1          :                                -> Sequencenumber
  succ       : Sequencenumber                 -> Sequencenumber
  _eq_, _ne_ : Sequencenumber,Sequencenumber -> Bool

eqns forall a,b : Sequencenumber
  ofsort Sequencenumber
    succ(0) = 1;
    succ(1) = 0;

  ofsort Bool
    0 eq 0 = true;
    1 eq 1 = true;
    0 eq 1 = false;
    1 eq 0 = false;
    0 ne 1 = true;
    1 ne 0 = true;
    0 ne 0 = false;
    1 ne 1 = false;

    (*a eq b = b eq a;
    a ne b = b ne a;
    a eq a = true;
    a ne a = false;*)

endtype (* Sequencenumber *)

type InresSpType is Boolean, ISDUType, DecNumb
sorts SP
opns
  ICONreq,ICONconf,IDISind,
  ICONind,ICONresp,IDISreq                :      -> SP
  IDATreq,IDATind                         : ISDU -> SP
  isICONreq,isICONconf,isIDISind,isIDATreq,
  isIDATind,isICONind,isICONresp,isIDISreq: SP   -> Bool
  data                                    : SP   -> ISDU
  map                                     : SP   -> DecNumb

eqns forall d : ISDU, sp : SP
  ofsort DecNumb
    map(ICONreq)    = 0;
    map(ICONconf)   = 1;
    map(IDISind)    = 2;
    map(IDATreq(d)) = 3;
    map(IDATind(d)) = 4;
    map(ICONind)    = 5;
    map(ICONresp)   = 6;
    map(IDISreq)    = 7;

  ofsort ISDU
    data(IDATreq(d)) = d;
    data(IDATind(d)) = d;

  ofsort Bool
    isICONreq(sp)  = map(sp) == 0;
    isICONconf(sp) = map(sp) == 1;
    isIDISind(sp)  = map(sp) == 2;
    isIDATreq(sp)  = map(sp) == 3;
    isIDATind(sp)  = map(sp) == 4;
    isICONind(sp)  = map(sp) == 5;
    isICONresp(sp) = map(sp) == 6;
    isIDISreq(sp)  = map(sp) == 7;
endtype (* InresSpType *)

type IPDUType is Boolean, ISDUType, DecNumb, Sequencenumber
sorts IPDU
opns
```



```
  CR,CC,DR           :                       -> IPDU
  DT                 : Sequencenumber,ISDU -> IPDU
  AK                 : Sequencenumber        -> IPDU
  isCR,isCC,isDT,
  isAK,isDR          : IPDU                  -> Bool
  data               : IPDU                  -> ISDU
  num                : IPDU                  -> Sequencenumber
  map                : IPDU                  -> DecNumb

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 Sequencenumber
    num(DT(f,d)) = f;
    num(AK(f)) = f;

  ofsort Bool
    isCR(ipdu) = map(ipdu) == 0;
    isCC(ipdu) = map(ipdu) == 1;
    isDT(ipdu) = map(ipdu) == 2;
    isAK(ipdu) = map(ipdu) == 3;
    isDR(ipdu) = map(ipdu) == 4;
endtype (* IPDUType *)

type MediumSpType is Boolean, IPDUType, DecNumb
sorts MSP
opns
  MDATreq,MDATind       : IPDU -> MSP
  isMDATreq,isMDATind   : MSP  -> Bool
  data                  : MSP  -> IPDU
  map                   : MSP  -> DecNumb

eqns forall d : IPDU, sp : MSP
  ofsort DecNumb
    map(MDATreq(d)) = 8;
    map(MDATind(d)) = 9;

  ofsort IPDU
    data(MDATreq(d)) = d;
    data(MDATind(d)) = d;

  ofsort Bool
    isMDATreq(sp) = map(sp) == 8;
    isMDATind(sp) = map(sp) == 9;
endtype (* 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,MSAP1] :noexit:=
    hide IPdu_ini in
                Initiator[ISAPini,IPdu_ini]
     |[IPdu_ini]| Coder[IPdu_ini,MSAP1]

    where

    process Initiator[ISAP,IPdu] :noexit:=
     (Connectionphase[ISAP,IPdu]
    >>Dataphase[ISAP,IPdu] (succ(0)))
    [>Disconnection[ISAP,IPdu]

      where

      process Connectionphase[ISAP,IPdu] :exit:=
        Connectrequest[ISAP,IPdu]
      >>accept z:DecNumb in Wait[ISAP,IPdu](z)
```

```
      where

      process Connectrequest[ISAP,IPdu] :exit(DecNumb):=
       (ISAP?sp:SP;([isICONreq(sp)]->IPdu!CR;exit(s(0))
                   [][not(isICONreq(sp))]->Connectrequest[ISAP,IPdu])
       (* User errors are ignored *)
      []IPdu?ipdu:IPDU[not(isDR(ipdu))];Connectrequest[ISAP,IPdu])
       (* DR is only accepted by process Disconnection *)
       (* System errors are ignored *)
      endproc (* Connectrequest *)

      process Wait[ISAP,IPdu](z:DecNumb) :exit:=
       (IPdu?ipdu:IPDU[not(isDR(ipdu))];([isCC(ipdu)]
                                          ->ISAP!ICONconf;exit
                                     [][not(isCC(ipdu))]
                                          ->Wait[ISAP,IPdu](z))
       (* DR is only accepted by process Disconnection *)
       (* System errors are ignored *)
      []i;([z < 4]->IPdu!CR;Wait[ISAP,IPdu](s(z))
         [][z == 4]->ISAP!IDISind;Connectionphase[ISAP,IPdu])
           (* Timeout *)
      []ISAP?sp:SP[not(isIDISind(sp))];Wait[ISAP,IPdu](z))
       (* User errors are ignored *)
      endproc (* Wait *)
    endproc (* Connectionphase *)

    process Dataphase[ISAP,IPdu](number:Sequencenumber) :noexit:=
      Readytosend[ISAP,IPdu](number)
      (* 1 is the first Sequencenumber *)
    >>accept z:DecNumb,number:Sequencenumber,olddata:ISDU
           in Sending[ISAP,IPdu](z,number,olddata)
      (* z is number of sendings. At the beginning z=1 *)
      where
      process Readytosend[ISAP,IPdu](number:Sequencenumber):
                  exit(DecNumb,Sequencenumber,ISDU):=
       (ISAP?sp:SP;
               ([isIDATreq(sp)]
                  ->IPdu!DT(number,Data(sp));exit(s(0),number,Data(sp))
              [][not(isIDATreq(sp))]->Readytosend[ISAP,IPdu](number))
      []IPdu?ipdu:IPDU[not(isDR(ipdu))];Readytosend[ISAP,IPdu](number))
      endproc (* Readytosend *)

    process Sending[ISAP,IPdu]
               (z:DecNumb,number:Sequencenumber,olddata:ISDU):noexit:=
     (IPdu?ipdu:IPDU[not(isDR(ipdu))];
           ([isAK(ipdu) and (num(ipdu) eq number)]
                                  ->Dataphase [ISAP,IPdu](succ(number))
          [][isAK(ipdu) and (num(ipdu) ne number)  and  (z < 4)]
                    ->IPdu!DT(number,olddata);
                               Sending[ISAP,IPdu](s(z),number,olddata)
     (* The Initiator shall not resend more than 4 times in case of *)
     (* faulty transmission *)
          [][isAK(ipdu) and (num(ipdu) ne number)  and  (z == 4)]
                    ->IPdu!DR;ISAP!IDISind;Initiator[ISAP,IPdu]
          [][not(isAK(ipdu))]->Sending[ISAP,IPdu](z,number,olddata))
    []i;([z < 4]->IPdu!DT(number,olddata);
                    Sending[ISAP,IPdu](s(z),number,olddata)
       [][z == 4]->ISAP!IDISind;Initiator[ISAP,IPdu])
    []ISAP?sp:SP[not(isIDATreq(sp))];
                    Sending[ISAP,IPdu](z,number,olddata))
    endproc (* Sending *)
  endproc (* Dataphase *)

  process Disconnection[ISAP,IPdu]:noexit:=
    IPdu!DR;ISAP!IDISind;Initiator[ISAP,IPdu]
  endproc (* Disconnection *)
endproc (* Initiator *)

process Coder[IPdu,MSAP] :noexit:=
 (IPdu?ipdu:IPDU;MSAP!MDATreq(ipdu);Coder[IPdu,MSAP]
[]MSAP?sp:MSP;IPdu!data(sp);Coder[IPdu,MSAP])
endproc (* Coder *)
endproc (* Station_Ini *)

process Station_Res[MSAP2,ISAPres] :noexit:=
hide IPdu_res in
                Responder[ISAPres,IPdu_res]
    |[IPdu_res]|Coder[IPdu_res,MSAP2]

where

process Responder[ISAP,IPdu]:noexit:=
 (Connectionphase[ISAP,IPdu]
>>Dataphase[ISAP,IPdu](succ(1)))
[>Disconnection[ISAP,IPdu]

  where

  process Connectionphase[ISAP,IPdu]:exit:=
    Connectrequest[ISAP,IPdu]
```



```
    >>Wait[ISAP,IPdu]

      where

      process Connectrequest[ISAP,IPdu]:exit:=
       (IPdu?ipdu:IPDU;([isCR(ipdu)]->ISAP!ICONind;exit
                      [][not(isCR(ipdu))]->Connectrequest[ISAP,IPdu])
           (* System errors are ignored *)
      []ISAP!ICONresp;Connectrequest[ISAP,IPdu])
           (* User errors are ignored *)
      endproc (* Connectrequest *)

      process Wait[ISAP,IPdu] :exit:=
       (IPdu?ipdu:IPDU;Wait[ISAP,IPdu]
           (* System errors are ignored *)
      []ISAP!ICONresp;IPdu!CC;exit)
      endproc (* Wait *)
    endproc (* Connectionphase *)

    process Dataphase[ISAP,IPdu](number:Sequencenumber):noexit:=
           (* number is the last acknowledged Sequencenumber *)
     (IPdu?ipdu:IPDU;([isDT(ipdu) and (num(ipdu) eq succ(number))]
                         ->ISAP!IDATind(data(ipdu));IPdu!AK(num(ipdu));
                              Dataphase[ISAP,IPdu](succ(number))
                    [][isDT(ipdu) and (num(ipdu) eq number)]
                         ->IPdu!AK(num(ipdu));
                              Dataphase[ISAP,IPdu](number)
                    [][isCR(ipdu)]->ISAP!ICONind;Wait[ISAP,IPdu]
                    [][not(isDT(ipdu) or isCR(ipdu))]
                         ->Dataphase[ISAP,IPdu](number))
    []ISAP!ICONresp;Dataphase[ISAP,IPdu](number))
         (* User errors are ignored *)
      where

      process Wait[ISAP,IPdu]:noexit:=
       (IPdu?ipdu:IPDU;Wait[ISAP,IPdu]
           (* System errors are ignored *)
      []ISAP!ICONresp;IPdu!CC;Dataphase[ISAP,IPdu](succ(1)))
      endproc (* Wait *)
    endproc (* Dataphase *)

    process Disconnection[ISAP,IPdu] :noexit:=
      ISAP!IDISreq;IPdu!DR;Responder[ISAP,IPdu]
    endproc (* Disconnection *)
  endproc (* Responder *)

  process Coder[IPdu,MSAP] :noexit:=
   (IPdu?ipdu:IPDU;MSAP!MDATreq(ipdu);Coder[IPdu,MSAP]
  []MSAP?sp:MSP;IPdu!data(sp);Coder[IPdu,MSAP])
  endproc (* Coder *)
endproc (* Station_Res *)
endspec
```

## 5. Experiences and evaluation

The specifications have been ckecked 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 [vEI88]. The sematics have been checked by performing a limited number of simulation experiences on the specification 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 Inres. 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**


[BHS91] Belina, F., Hogrefe, D., Sarma, A.: SDL with applications from protocol specification, Prentice-Hall, 1991

[BHT88] Belina, F., Hogrefe, D., Trigila, S.: Modelling OSI in SDL (in Turner: Formal Description Techniques, North-Holland, Amsterdam, 1988)

[BRO87] Broy, M. et al: A stream function definition of MASCOT. System Designers, Software Technology Centre, Final Report, 1987.

[CHA87] Chan, I.: Estelle-C compiler, Version 2.0, University of British Columbia, 1987.

[GOT91] Gotzhein, R.: Specifying Communication Services with Temporal Logic (in Logrippo, L. et al (eds.): Protocol specification, testing and verification X), North-Holland, 1991.

[HOG89] Hogrefe, D.: Estelle, LOTOS und SDL, Springer Verlag, 1989.

[ISO 7498] ISO TC97/SC21: Basic Reference Model (ISO/IS 7498, 1984)

[ISO 8807] ISO TC97/SC21: LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour (ISO/IS 8807, 1988)

[ISO 9074] ISO TC97/SC21: Estelle - A formal description technique based on an extended state transition model (ISO/IS 8807, 1988)

[SPI89] Spivey, J.M.: The Z notation, Prentice-Hall, 1989.

[TR 8509] ISO TC97/SC21: OSI Service Conventions ( ISO/TR 8509, 1987)

[TR 10167] ISO TC97/SC21: Guidelines for the application of Estelle, LOTOS and SDL (ISO TR 10167, 1990)

[vEI88] van Eijk, P.: Software tools for the specification language LOTOS, Twente University, 1988.

[VSS88] Vissers, C., Scollo, G., van Sinderen, M.: Architecture and specification style in formal descriptions of distributed systems (in Aggarwal, S., Sabnani, K.: Protocol specification, testing and verification VIII), North Holland, 1988

[Z100] CCITT Recommendation Z.100: Specification and Description Language SDL (Blue Book, Volume X.1 - X.5, 1988, ITU General Secretariat - Sales Section, Places des Nations, CH-1211 Geneva 20)