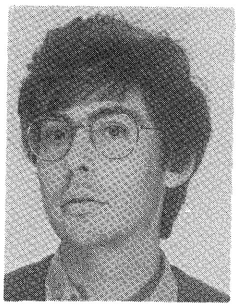
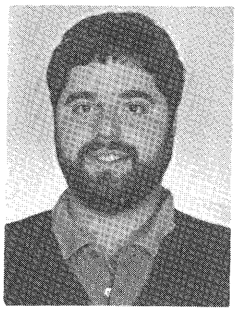
Study of the resynchronization of a communication protocol

Bernard COUSIN, Pascal ESTRAILLIER

Université Pierre et Marie Curie, Laboratoire MASI, Tour 65–66, 4 place Jussieu, 75252 Paris Cedex 05, France



Bernard Cousin, who holds a Doctorate from the University of Paris VI, lectures at the Programming Institute of the University of Paris VI. He works in the 'modelling and parallelism' team of the MASI laboratory, specializing on the integration of the specification, modelling and validation phases for distributed systems



Pascal Estraillier, who holds a Doctorate from the University of Paris VI, lectures at the Programming Institute. He is a member of the 'modelling and parallelism' team of the MASI laboratory, where he is involved in research on design, modelling and validation of robust communication protocols.

COMMENTARY

The modelling of communication protocols forms a particular field of application for any theory of parallelism. This is due, essentially, to the way in which data are handled in this class of application. Data travelling through a communication medium are subject both to heavy synchronization constraints and to minor transformations.
Over a number of years now, researchers such as G. Bethelot, M. Diaz and C. Girault have demonstrated the value of the use of formal models such as those based on Petri nets. This article by B. Cousin and P. Estraillier falls within this framework and shows how precisely predicate nets make it possible to describe and analyse complex situations.
If the modelling of a system remains part of the 'engineer's art', this type of analysis could, in the years to come, be assisted by computers. Major projects are developing in this direction today, in the United States and in Europe, within the framework of Race or Esprit. In their conclusion, the authors mention the Estelle and Lotos languages, a clear reference to the Esprit SEDOS projects, due to be completed in 1987, and which we hope to discuss in greater length in these columns later.

Gerard Memmi

0264–7419/87/060461–11$05.50

CONTENTS

1. Introduction

Much research work is taking place on the modelling of communication protocols [Berthelot 83, Azema 85, Estraillier 86]. The complexity of their operating mechanisms makes it difficult and often cumbersome to build up modes of parallel systems, since such models require detailed and complete studies of the mechanisms used. Modelling must incorporate all the (often very numerous) functions implemented. A partial analysis of these systems and the introduction of simplifying assumptions are unlikely to be satisfactory unless the simplifications can be fully justified.

In this article, we propose an original approach to modelling, without attempting an exhaustive analysis of the internal mechanisms implemented, but based instead on methodic construction of abstractions allowing incorporation of various functionalities of the system studied. This method highlights the properties of the model and allowing functional validation of the properties of the system. It is particularly appropriate for a study of systems structured in hierarchical layers, as it flows directly from work on standardization of telecommunication protocols [Iso-7498 83]. Our approach sets out to implement a local study of each of the layers taking into account, when characterizing its execution environment, only interfaces with adjacent layers. This kind of study is of fundamental importance when examining complex systems, since the system environment is completely represented (i.e. without making simplifying assumptions) and in a concise way.

We apply this approach to modelling, by means of predicate Petri nets [Brams 82], of services provided by the network layer (ISO level 3) during the data transfer stage. We are particularly interested in handling breakdowns on network stations, and in describing the resynchronization mechanism for transmission. We therefore provide a complete model of the loss of data packets following the breakdown of the station, the detection of this event and the transmission by the network layer of particular packets (reinitialization packets) allowing the transport layer to be warned of a failure.

After describing the various steps in our approach, we describe our model of the services provided by the network layer for data transfer [Iso-8348 84]. We then validate this model functionally by showing that the properties corresponding to it conform to the properties defining a service.

Such a model can then be used for studies of the upper layer (Transport [Iso-8072 84, Iso-8073 84]) which also incorporates, in a very reduced form, the services of the network layer [Cousin 87]. In this way, our study provides a precise and complete description of protocols without going into implementation details.

2. The functional validation approach

Any study of a system can give rise to several different models differing in their form, accuracy and their fidelity to the system being modelled. To validate the model proposed, it is therefore necessary to evaluate its conformity to the system description. The approach that we propose is divided into four stages (Fig. 1).

1. *Specification of the properties of the system* that we are setting out to model, based on a study of the way it operates. Such a study will identify external properties characterizing its operations.
2. *Modelling of external mechanisms* forming the system, in order to obtain a model which is an abstraction of the system characteristics and not a description of its internal mechanisms. In this way we can build up a model independent of any particular implementation, since it is based only on the properties of the system.

   For our model, we use predicate nets which are a restricted form of Petri nets. This is a good tool for producing concise behavioural descriptions, of complex parallel systems. Note, however, that our approach does not require any particular type of model, and a choice can be made based on the properties to validate.
3. *Analysis of the model* makes it possible to test that it has the required properties.

   In this example, we principally use the assertions method to check the properties of the model. The principle underlying this method is to check that a statement is satisfied for every state of the model. Here again, there is no obligation to select a

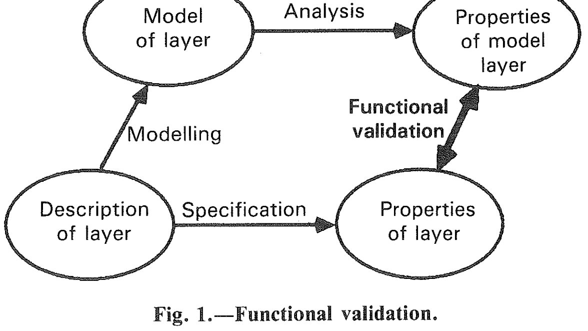


Fig. 1.—Functional validation.

particular process for determining the properties of the model. Properties can be selected according to the types of results required and the complexity of the model.

4. *Functional validation of the model* shows whether or not the properties of the model correspond to those of the system being studied. This stage gives a formal demonstration that the properties obtained in the model analysis stage imply the system properties specified in the first stage. The model emerging from the second stage is therefore an equivalent abstraction, from the functional point of view, of the system studied.

In the framework of a hierarchical system, and in particular of a communication system, or approach allows local study of the behaviour of any layer, to give a model of it that has been functionally validated and is therefore reusable for an overall study.

3. Network service

Standardization in computer interconnection is based on a seven-layer architecture [Zimmermann 80]. In this architecture, each layer is described by distinguishing a *protocol*, which specifies message exchange mechanisms, and a *service* which characterizes the functions provided to the layer above.

We start by describing the service provided by the network layer for handling breakdowns during the data transfer stage, and then characterize it by means of four properties. Our study will be based on the standard for networks of the TRANSPAC type [Transpac 79].

3.1. Description of the network service

During data transfer on a network connection (called the virtual circuit), the network layer establishes a two-way link between two subscribers. It permits despatch, over the virtual circuit, of data structured into packets. The two paths on the virtual circuit are independent.

The network service, as opposed to the network protocol, is simply an external view of the functionalities of the network layer. Consequently, it gives no description of internal mechanisms (routing, retransmitting, window management, etc.).

At the level of the virtual circuit, several types of situations may occur;

(a) *Normal operations:*
On each pathway, packets transmitted by a subscriber are taken over by the network service and, having moved over the virtual circuit, are delivered to the destination subscriber.

(b) *Operation in a failing environment:*
During the transfer stage, an incident (station breakdown, inconsistent state of the protocol, etc.) may perturb packet transfer (loss, degradation, etc.). Three stages can then be observed:

(i) *Desynchronization:* An element on the virtual circuit fails; it is then desynchronized with respect to other elements. Packets which it is handling may then contain inconsistent information.

(ii) *Loss:* The incorrect operation of the desynchronized element leads to loss of particular packets.

(iii) *Resynchronization:* The network service ensures that desynchronization is followed by resynchronization. This stage allows the two subscribers to be warned of the occurrence of a failure. It involves transmitting a reinitialization packet on each of the communication paths. This particular type of packet is handled, at the level of the two subscribers, by the layer above (the transport layer) which, having been warned of the incident, sets in motion a restart procedure.

Note, however, that reinitialization packets may also be lost due to a further desynchronization. Such a loss is not very damaging since it leads to the regeneration of a new reinitialization packet on both communication paths.

3.2. Interface with the transport layer

At each layer, particular processes (called *entities*) handle all the operations providing the service for the layer. This section describes the actions executed by these entities for data transfer management.

Consider two subscribers at the ends of the virtual circuit. Each path has a transmitting end and a receiving end. Data transfer between these two ends takes place as follows:

1. The entity belonging to the transport layer at the transmitting end issues a transmission request to the local network entity.
2. The network entity at the transmitting end uses a network protocol to transmit the data (structured as a packet) to the network entity at the other end.
3. The network entity at the receiving end tells the transport entity that a packet has been received.

The interfaces between the transport and network layers can therefore be reduced to the following three primitives:

1. *N-SDU-Data-Request:* The transport layer sends a data transmission request to the network layer.
2. *N-SDU-Data-Indication:* The network layer tells the transport layer that a data packet has been received.
3. *N-SDU-Reset-Indication:* the network layer tells the transport layer that it has received a reinitialization packet.

The other primitives defined by the standard (notably, N-SDU-RESET-req) are not involved in the data transfer stage of the network service.

3.3. Properties of the service

The following four properties sum up the service provided by the network layer in the face of desynchronizations during data transfer:

SP0: Packets can always be transferred (no irrepairable blockage takes place).

SP1: The order of packets is preserved during the transfer stage (the order of transmission of packets corresponds to their order of reception).

SP2: The transfer phase will not duplicate packets (every data packet is either transmitted or lost).

SP3: To handle desynchronizations, the network service guarantees:

(i) A reinitialization packet is transmitted to each end of the virtual circuit after any desynchronization.

(ii) Packets submitted to the network *before* reception of a reinitialization packet are either sent to their destination, or deleted.

(iii) Packets submitted to the network *after* reception of reinitialization packet are delivered to the receiving end once it has received the reinitialization packet.

These four properties of the network service should, in our approach, be implied by the properties resulting from analysis of the model proposed in the preceding section. Our model should therefore be functionally equivalent to a model of the protocol used by the network layer.

4. Model

In this section, we define the structures to model the service provided by the network layer for data transfer. We then give a brief review of predicate Petri nets, before describing our model.

4.1. Service representation structures

Our task, taking the network specification as starting point, is to build up a model whose properties correspond to those of the service we wish to model. To do so, we propose a structure on which a functional representation of the service will be built. This structure is not used to describe operations actually carried out by the network layer (i.e. the protocol) but allows construction of the model for validation.

This section considers the structures on which the model of the network service will be based. We describe information managed by the transport layer, and then describe the solution adopted to represent the virtual circuit.

4.1.1. *Representation of the transport layer*

The network service only perceives the transport layer in the form of packets transferred from one subscriber to the other. As information contained in packets is transparent to the network layer, there is no need to represent it in our study. However, certain properties of the network service (**SP1** concerning preservation of order and **SP2** concerning avoidance of duplication) require some knowledge of the information. To validate our model, it was therefore necessary to identify packets transmitted by the transport entities corresponding to the transmitting ends of the circuit; we decided to number them. We therefore defined a *counter* for each of the transmission pathways. These counters are built into the model to characterize the environment of our representation of the network service. They are only used to validate the model without disturbing its operations. Counters are initialized to zero. Every time a transmission is sent over a transmission path, the corresponding counter is incremented.

Characteristics of a data item for transfer

For our model, the data item contained in a packet is assimilated to the number assigned to the packet. A data item transmitted at the level of the transport layer by one end of the circuit can then be characterized by the pair ⟨ **data, direction** ⟩ where:

1. **data:** value of the transmission counter.
2. **direction:** transmission path represented.

4.1.2. *Representation of the virtual circuit*

The virtual circuit is formed of several elements (stations, connection devices, etc.) which contribute to forwarding packets. Functionally, there is no need to distinguish these various elements. We can then see each path in the virtual circuit as a set of locations, each of which corresponds to a packet storage zone. A location can therefore be empty or contain a packet. To represent the chronology of packet arrivals on a communication pathway, this set has to be organized into a queue. The size of the queue (QUEUESIZE) corresponds to the maximum number of packets that can be travelling simultaneously over the connection. The first location (number 1) is therefore associated with the transmitting end and the last (number QUEUESIZE) is associated with the receiving end. Intermediate locations correspond to packets being transferred.

Figure 2 shows the two queues used to represent the two pathways in the virtual circuit. An element of each circuit is represented using a pair of locations, each of which belongs to a different queue. On each pathway, locations are numbered in increasing order starting from the transmitting end. The numbers corresponding to the locations in a pair are therefore complementary.

Location management

The operations carried out on locations allow a representation of actions carried out by the network service on packets. The nature of these operations depends on the state of the circuit. We shall therefore return to the situations described in section 3.1. and interpret them from the point of view of location management.

(a) *Normal operations*
The two queues are managed independently. Management of the location merely means organizing the movement of packets, ensuring that they travel through the queue.
Transit: for a packet to be transferred to a location, that location must be empty. By virtue of the way the queue is constructed, the packet to store comes from the location with immediately lower number.

(b) *Operation in a failing environment*
To integrate the failed situation into the model, we have to manage the pair of locations representing an element of the virtual circuit.
Desynchronization: locations belonging to the pair

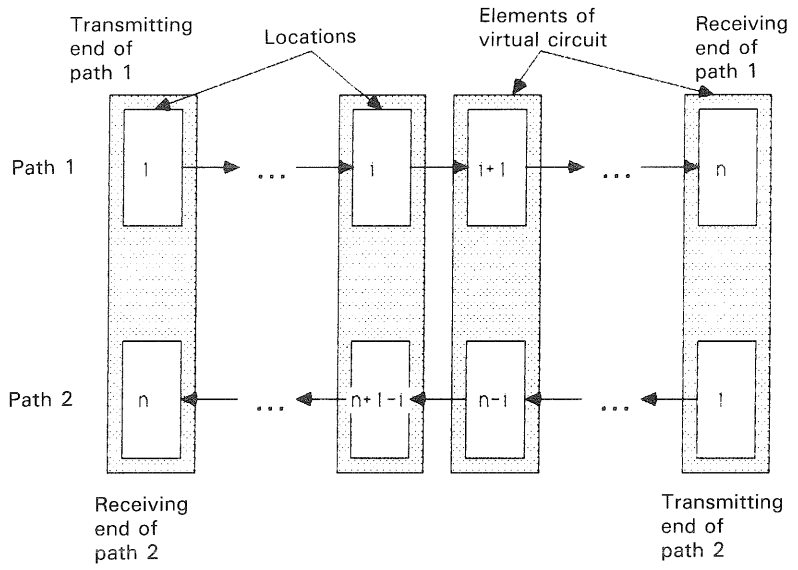


Fig. 2.—Representation of the virtual circuit.

associated with a failed element become desynchronized. The packets contained in these locations are not damaged.

Loss: the packet contained in a desynchronized location is lost. The location becomes empty. At the level of a pair of locations, the loss is observed independently.

Resynchronization: the locations belonging to a pair involved in failure handling are synchronized again. Processing involves generating reinitialization packets in these locations.

After this procedure, locations are again in a normal state (synchronized). The synchronization packet then moves through the queue in the same way as a data packet.

Characteristics of a location

A location can be defined by a quadruplet ⟨**no,info, state,direction**⟩ so as to conform with all the operations affecting a location, as follows:

(i)	**no:**	location number (the rank number in the queue);
(ii)	**info:**	information contained in the location (nothing or a data packet or reinitialization packet);
(iii)	**state:**	state of the location (synchronized or desynchronized);
(iv)	**direction:**	transmission pathway represented

4.2. Predicate Petri nets

Predicate Petri nets are an abbreviation of traditional Petri nets. They allow complex systems to be modelled in a particularly concise form. Our definition is based on that given in [Genrich 79].

A predicate Petri net is a sextuplet ⟨**P**, **T**, **C**, **V**, **A**, **L**⟩ such that:

(i) **P** is a finite set of places.
(ii) **T** is a finite set of transitions.
(iii) **C** is a finite set of constants.
(iv) **V** is a finite set of variables with values drawn from **C**.
(v) **A** is a set of arcs forming a two-part graph between **P** and **T**; arcs are labelled by the formal sums of the *n*-tuplets formed from variables of **V**.
(vi) **L** is a set of labels which are logical expressions in variables **V**.

To cross a transition in the network, we have to substitute each occurrence of a variable **V** on one of the arcs linked to a transition with the same value of **C**.

(i) The transition is firable if the logical expression associated with the transition is true after substitution and if the entry places contain sufficient occurrences of the constants as required by valuations of the entry arcs to the transition.
(ii) Firing of a transition removes the entry places from the transition and adds to the output places as many occurrences of constants as appear in the valuations of the output arcs from the transition.

4.3. Description of the model

The representation power of predicate Petri nets allows a particularly simple model to be proposed (Fig. 3), since all the elements to model have been reduced to the level of the representation of transition pathways, the state of elements in the circuit and of locations.

The structure of the model clearly separates elements belonging to the network layer, from those that are associated with the transport level and those which are situated in the interface.

4.3.1. *Elements modelling the transport layer*

In the transport layer, it is only useful to represent the generation of data by the transmitting ends of each transmission pathway.

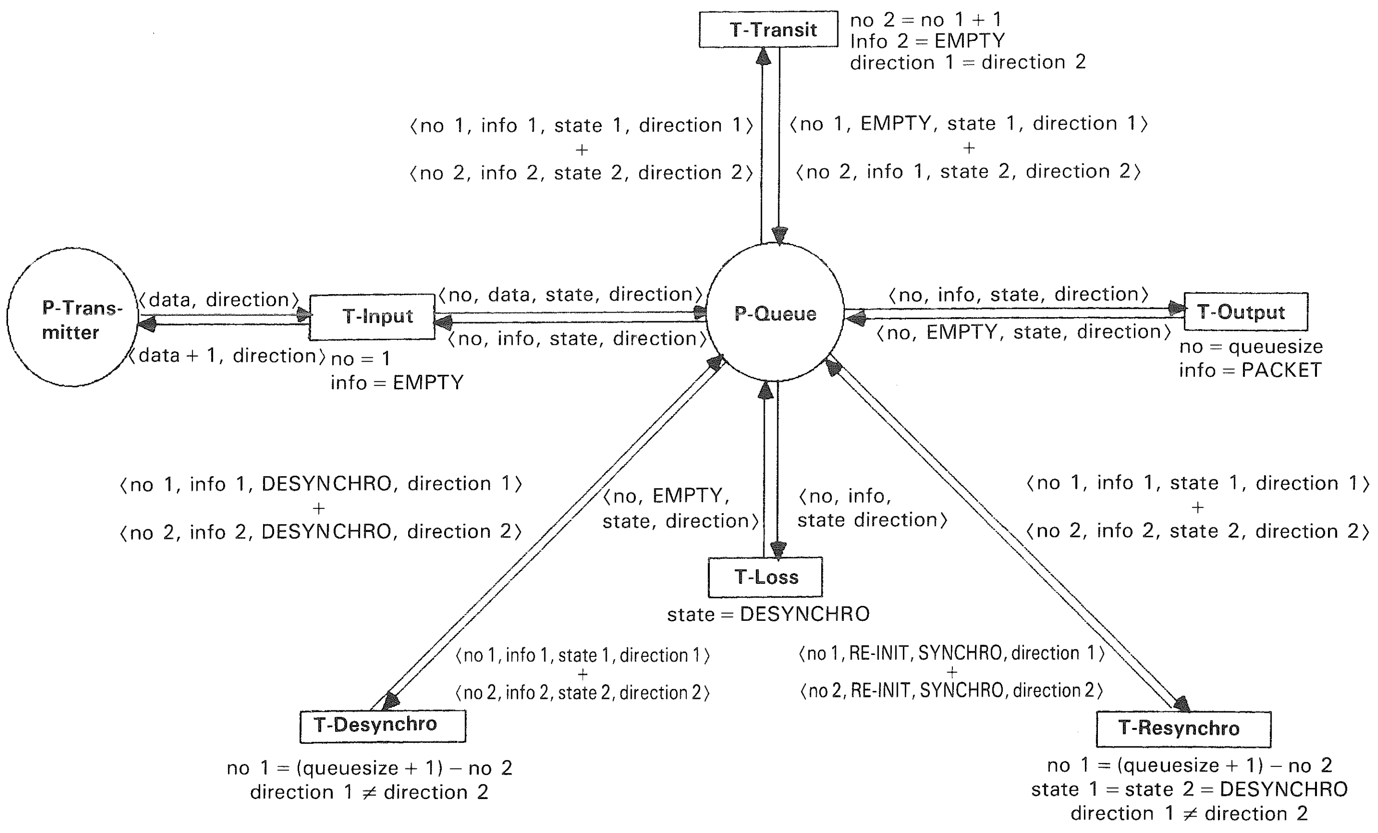

T-Transit
no 2 = no 1 + 1
Info 2 = EMPTY
direction 1 = direction 2
⟨no 1, info 1, state 1, direction 1⟩ + ⟨no 2, info 2, state 2, direction 2⟩
⟨no 1, EMPTY, state 1, direction 1⟩ + ⟨no 2, info 1, state 2, direction 2⟩
P-Transmitter
⟨data, direction⟩
⟨data + 1, direction⟩
T-Input
no = 1
info = EMPTY
⟨no, data, state, direction⟩
⟨no, info, state, direction⟩
P-Queue
⟨no, info, state, direction⟩
⟨no, EMPTY, state, direction⟩
T-Output
no = queuesize
info = PACKET
⟨no 1, info 1, DESYNCHRO, direction 1⟩ + ⟨no 2, info 2, DESYNCHRO, direction 2⟩
⟨no, EMPTY, state, direction⟩
⟨no, info, state direction⟩
T-Loss
state = DESYNCHRO
⟨no 1, info 1, state 1, direction 1⟩ + ⟨no 2, info 2, state 2, direction 2⟩
⟨no 1, info 1, state 1, direction 1⟩ + ⟨no 2, info 2, state 2, direction 2⟩
⟨no 1, RE-INIT, SYNCHRO, direction 1⟩ + ⟨no 2, RE-INIT, SYNCHRO, direction 2⟩
T-Desynchro
no 1 = (queuesize + 1) − no 2
direction 1 ≠ direction 2
T-Resynchro
no 1 = (queuesize + 1) − no 2
state 1 = state 2 = DESYNCHRO
direction 1 ≠ direction 2

Fig. 3.—Model of the network service.

Place **P-Transmitter:** This place models the transmitting ends of the virtual circuit. It contains marks defined by the pair ⟨**data, direction**⟩ such that:

1. **data** ∈ ℕ
2. **direction** ∈ {PATH1, PATH2}

4.3.2. *Elements modelling the network layer*

We have to represent locations belonging to two queues, by presenting the actions which modify them.

Place **P-Queue:** This place models the two queues represented in the virtual circuit. It therefore contains as many marks as there are locations in the queues (i.e. **QUEUESIZE***2).

Each mark is defined by the quadruplet ⟨**no, info, state, direction**⟩ introduced in the preceding section. The domain of each component is the following:

- **no** ∈ {1...**QUEUESIZE**}
- **info** ∈ {**EMPTY, packet**} such that

  packet ∈ {**RE − INIT, data**} and **data** ∈ ℕ

- **state** ∈ {**SYNCHRO, DESYNCHRO**}
- **direction** ∈ {**PATH1, PATH2**}

Transition **T-Transit:** This transition models the movement of packets. It requires two markers u_1 = ⟨**no**$_1$, **info**$_1$, **state**$_1$, **direction**$_1$⟩ and u_2 = ⟨**no**$_2$, **info**$_2$, **state**$_2$, **direction**$_2$⟩ in place **P-Queue** such that:

On input: **no**$_2$ = **no**$_1$ + 1 (i.e. u_2 is the location following u_1)
- **info**$_2$ = **EMPTY** (i.e. u_2 contains no information)
- **direction**$_1$ = **direction**$_2$ (i.e. u_1 and u_2 correspond to the same path).

On output: **info**$_1$:= **EMPTY** (i.e. u_1 no longer contains information)
- **info**$_2$:= **info**$_1$ (i.e. the information in u_1 has been transferred to u_2)

Transcription **T-Desynchro:** This transition models desynchronization between a pair of locations. It requires two markers u_1 = ⟨**no**$_1$, **info**$_1$, **state**$_1$, **direction**$_1$⟩ and u_2 = ⟨**no**$_2$, **info**$_2$, **state**$_2$, **direction**$_2$⟩ in the **P-Queue** place such that:

On input: **no**$_1$ = (**QUEUESIZE** + 1) − **no**$_2$ (i.e. u_1 and u_2 form a pair)
- **direction**$_1$ ≠ **direction**$_2$ (i.e. u_1 and u_2 do not correspond to the same path)

On output: **state**$_1$:= **state**$_2$:= **DESYNCHRO** (i.e. u_1 and u_2 are desynchronized).

Transition **T-Lost:** This transition models loss of packets. It requires a marker u = ⟨**no, info, state, direction**⟩ in place **P-Queue** such that:

On input: **state** = **DESYNCHRO** (i.e. u is desynchronized)
On output: **info** := **EMPTY** (i.e. u contains no information)

Transition **T-Resynchro:** This transition models resynchronization of a pair of locations. It implies two markers u_1 = ⟨**no**$_1$, **info**$_1$, **state**$_1$, **direction**$_1$⟩ and u_2 = ⟨**no**$_2$, **info**$_2$, **state**$_2$, **direction**$_2$⟩ in place **P-Queue** such that:

On input: **no**$_1$ = (**QUEUESIZE** + 1) − **no**$_2$ (i.e. u_1 and u_2 form a pair)
- **direction**$_1$ ≠ **direction**$_2$ (i.e. u_1 and u_2 do not correspond to the same path)
- **state**$_1$ = **state**$_2$ = **DESYNCHRO** (i.e. u_1 and u_2 are desynchronized)

On output: **state**$_1$:= **state**$_2$:= **SYNCHRO** (i.e. u_1 and u_2 are resynchronized)
- **info**$_1$:= **info**$_2$:= **RE − INIT** (i.e. the reinitialization packet is generated in each location).

4.3.3. *Elements modelling the interface*

The interface between the network and transport layers is modelled using transitions representing the primitives used.

Transition **T-Input:** This transition models a request for data transmission on a transmission pathway. It implies, on the one hand, a marker $u = \langle$**no, info, state, direction**$\rangle$ contained in **P-Queue**, and on the other hand, a marker $v = \langle$**data, direction**$\rangle$ contained in **P-Transmitter**.
The equality, at the marker level, of the **direction** components determines the pathway used.

On input: **no** = 1 (i.e. the deposit takes place in the first location)
- **info** = **EMPTY** (i.e. the location is available)

On output: **info** := **data** (i.e. the location contains the data)
- **data** := **data** + 1 (i.e. the transmission counter has been incremented)

Transition **T-Output:** This transition models the data packet reception or reinitialization indications. It requires a marker $u = \langle$**no, info, state, direction**$\rangle$ contained in **P-Queue**.

On input: **no** = **QUEUESIZE** (i.e. we consider the last location in the queue, corresponding to the receiving end)
- **Info** = **packet** (i.e. the location is not empty)

On output: **Info** := **EMPTY** (i.e. the location contains the data)

4.4. Initial marking

It is helpful to introduce a notation allowing easy handling of the tuplets in the model. This notation will be used to define the initial marking and in the valuation of the model. We define a set of elementary projections making it possible to reach the component of a tuplet using simple functions.

Consider the tuplet u_i such that $u_i = \langle$**no**$_i$, **info**$_i$, **state**$_i$, **direction**$_i\rangle$.

- locno(u_i): returns the number of the location (**no**$_i$) associated with u_i.
- state(u_i): returns the state (**state**$_i$) associated with u_i.
- direction (u_i): returns the transmission path (**direction**$_i$) associated with u_i.

To handle the component **info**$_i$, we define two complex functions:

- packettype(u_i): returns the type of packet associated with tuplet u_i,
```
  packettype(u_i) := if info_i = EMPTY then EMPTY
                     else if info_i = RE - INIT then
                     RE - INIT
                     else DATA;
```
- datano(u_i): returns the number of the packet associated with tuplet u_i
```
  datano(u_i) := if packettype (u_i) = DATA then info_i
                 else UNDETERMINED
```

The initial marking corresponds to the state of the system after the establishment of the virtual circuit. We write $M_0(P)$ for all the markers contained in place P in the initial state.

Place **P-Transmitter:** In the initial state, the counter associated with each transmission way contains the value zero.

$$M_0(\textbf{P-Transmitter}) = \{\langle 0, \textbf{PATH1}\rangle ; \langle 0, \textbf{PATH2}\rangle\}$$

Place **P-Queue:** In the initial state, all the locations are empty and synchronized. Their location number and their pathways allow them to be distinguished.

- locations are empty:

$$\forall u_i \in M_0(\textbf{P-Queue});\ \text{packettype}\ (u_i) = \textbf{EMPTY}$$

- locations are synchronized:

$$\forall u_i \in M_0(\textbf{P-Queue});\ \text{state}\ (u_i) = \textbf{SYNCHRO}$$

- locations are clearly identified:

$$\{\text{locno}(u_i)/u_i \in M_0(\textbf{P-Queue})\ \text{and direction}(u_i) = \textbf{PATH1}\} = [1,\textbf{QUEUESIZE}]$$
$$\{\text{locno}(u_i)/u_i \in M_0(\textbf{P-Queue})\ \text{and direction}(u_i) = \textbf{PATH2}\} = [1,\textbf{QUEUESIZE}]$$

5. Evaluation of the model

Our service model for the network layer needs to be validated, i.e. it has to be shown that it respects the service properties described in section 3.3.

Below, we prove the theorems establishing these properties. We then go on to evaluate the conformity of the service model by showing the correspondence between properties of the model and those of the service. We can then conclude that there is a functional equivalence between our model of the network service and a model of the network protocol.

5.1. Properties of the model

To validate our model functionally, we have to determine the properties that it must satisfy.

MP0: The model is lively (there are no blocks). This property corresponds to the usual concept of liveliness.
MP1: When it comes to tuplet management, the model respects the ordering relation defined on packet numbers (it preserves the packet sequence).
MP2: The model will not manage more than one tuplet referencing the same packet (it does not duplicate packets).
MP3: Any crossing sequence describing the loss of a packet is extended by a sequence representing a resynchronization.

5.2. Checking the properties of the model

The properties of the model are demonstrated by means of three theorems. This section describes the proof method used; complete proofs are given in [Cousin 85].

For our proofs, we have to introduce the following sets:

1. **A:** All markings accessible from the initial marking.
2. **T:** All transitions in the model

It is also necessary to introduce functions for handling tuplets.

Let **P** be a set of tuplets contained in the **P-Queue** place.

Next (u_i, P): This function returns the tuplet (belonging to set P) corresponding to the first non-empty location whose number is less than that of the location named by u_i.
Let $u_i \in \mathbf{P}$, $u_k \in \mathbf{P}$; next $(u_i\mathbf{P}) = u_k$ iff:

- packettype(u_k) $\neq$ **EMPTY** (i.e. u_k contains a packet)
- locno(u_k) $\leqslant$ locno(u_i) (i.e. u_k is closer to the transmitter)
- $\forall u_n \in \mathbf{P}$ such that packettype(u_n) $\neq$ **EMPTY** and locno(u_n) $\leqslant$ locno(u_i) locno(u_n) $\leqslant$ locno(u_k) (i.e. u_k is between u_n and u_i)

First(**P**): This function returns the tuplet corresponding to the location (in the set specified) containing the first packet of data taken into account. This location is then the closest to the receiving end of all the non-empty locations in the set.

Let $u_k \in \mathbf{P}$; First(**P**) = u_k iff:

- packettype(u_k) ≠ **EMPTY**
- $\forall u_n \in \mathbf{P}$ such that packettype u_n ≠ **EMPTY** locno(u_k) ⩾ locno(u_n) (i.e. u_k is the closest to the receiving end)

Last(**P**): This function returns the tuplet corresponding to the location (from the specified set) containing the last packet of data taken into account.

Let $u_k \in \mathbf{P}$; Last(**P**) = u_k iff:

- packettype(u_k) = **DATA** (i.e. u_k contains a data package)
- $\forall u_n \in \mathbf{P}$ such that packettype (u_n) = **DATA** locno(u_k) ⩽ locno(u_n) (i.e. u_k is the closest to the transmitting end)

Path1(**P**): This function returns the set of tuplets belong to transmission pathway **PATH1** in set **P**.

Path1(**P**) = {$u_i \in \mathbf{P}$ such that direction(u_i) = **PATH1**}

Path2(**P**): This function returns the set of tuplets belonging to transmission pathway **PATH2** from set **P**.

Path2(**P**) = {$u_i \in \mathbf{P}$ such that direction (u_i) = **PATH2**}

Counter(**P**): This function returns the value of the component ⟨**data**⟩ of a mark contained in place **P-Transmitter**. The marker is determined according to the pathway associated with tuplets in set **P**.

5.2.1. *Proof of property* MPO

To prove property **MPO**, we use the following theorem:

Theorem TO: the model is lively (any transition in the model can always be made possible).

$$\forall M \in \mathbf{A}, \quad \forall t \in \mathbf{T}, \quad \exists S \in \mathbf{T}^* \quad \text{such that} \quad M(St\rangle$$

We prove the liveliness of the model in two stages. The first shows that the model has a set S of reception states. The second stage proves that in any state of S the model is quasi-lively.

Stage 1: The model has a set S of reception states. We define the set S by the following marking:

1. Tuplets of **P-Queue** are empty, and their state is synchronous.

   $\forall M \in \mathbf{E}$, $\forall u \in M$(**P-Queue**);

   packettype (u) = **EMPTY**, state(u) = **SYNCHRO**

2. The tuplets in **P-Transmitter** can be of any type.

   To prove that S is a reception set, we have to show that starting from the set **A** of accessible states, it is always possible to reach one of the states of S. According to [Keller 76] a model has a reception state S, if applying a norm **B** which assigns a weighting **N** to each marking:

   (i) the norm is zero for the reception state (**B**(S) = 0); and
   (ii) for every non-zero accessible marking, there exists a firing sequence which makes the norm decrease ($\forall M \in \mathbf{A}$ if $\mathbf{B}(M) \neq 0$ then

   $$\exists S \in T^* \text{ such that } M(S) > M' \text{ and } \mathbf{B}(M) > \mathbf{B}(M')).$$

We define our norm as follows:

$$\forall M \in \mathbf{A}, \ \forall u \in M(P\text{-Transmitter})\mathbf{B}(u) = 0;$$

$$\forall u \in M(P\text{-Queue})$$

$$\mathbf{B}(u) = \mathbf{B}(\text{locno}(u))^*(\mathbf{B}(\text{packettype}(u)) + \mathbf{B}(\text{state}(u))$$

with

$$\mathbf{B}(\text{no}) = \text{no}; \ \mathbf{B};(\mathbf{PACKET}) = 1; \quad \mathbf{B}(\mathbf{EMPTY}) = 0 \quad \mathbf{B}(\mathbf{DESYNCHRO}) = 2; \quad \mathbf{B}(\mathbf{SYNCHRO}) = 0.$$

Note that in the initial state the norm is zero.

The rest of the demonstration takes place in two sub-stages: we show that the model allows, on the one hand, resynchronization of all the stations, and, on the other hand, movement of data packets and reinitialization.

Sub-stage 1: It is possible to resynchronize desynchronized locations from place **P-Quene**.

The idea is to eliminate desynchronized locations from place **P-Queue**. We therefore require:

$\forall u \in M$(**P-Queue**); state(u) = **SYNCHRO**

For the demonstration, we need the following lemma:

Lemma L0: The paired tuplets for the representation of an element in a virtual circuit are always in the same state.

$\forall M \in \mathbf{A}$, $\forall u_i \in$ Path1(M(**P-Queue**)),

$\forall$ $u_k \in$ Path2(M(**P-Queue**)),
if locno u_i = (**QUEUESIZE** + 1) − *locno*(u_k) then
state(u_i) = state(u_k).

According to this lemma, tuplets associated with each of the two queues can be considered separately, as they always have the same state.

Let *path* be the set of tuplets in **P-Queue** belonging to one or other of the transmission pathways.

```
While ∃u ∈ Path; state (u) ≠ SYNCHRO (P-Queue contains at least one packet)
    Trigger T-Resynchro (a packet RE-INIT is generated in the location
                         which returns to the SYNCHRO state. The net-
                         work then contains at least one desynchronized
                         location).
End
```

Sub-stage 2: It is possible to provide the receiver with all the packets present in place **P-Queue** (the locations then become empty).

The idea is to empty locations in place **P-Queue.** We therefore require:

$\forall u \in M$(**P-Queue**); packettype(u) = **EMPTY**

To do so, we can consider the tuplets associated with each of the queues separately. All the transitions that have to be fired (**T-Transit** and **T-Output**) affect only one transmission path. We therefore have the following actions to carry out:

```
While ∃p ∈ M(P-Queue); p = First (M(P-Queue)) (P-Queue contains
                                                at least one packet)
    While locno(p) ≠ QUEUESIZE (i.e. p does not correspond to the
                                receiving end)
```

- Trigger **T-Transit** (this is always possible, since by their construction locations between locno(p) and **QUEUESIZE** are empty. The packet moves on one location)
- p = First(M(**P-Queue**)) (the packet nonetheless retains its status as First)

End
Trigger **T-Output** (now possible since conditions are satisfied. One packet at least is present on the network).
End

We have shown that the actions carried out lead to our obtaining tuplets characterizing empty and synchronized locations, which shows that set **S** is indeed a reception set. It is now necessary to prove that the actions described can always be executed, by showing that the model is quasi-lively.

Stage 2: In any state of the reception set of S, the model is quasi-lively.

To prove this property, we have to cross all the transitions in the model in turn.

In any state of the set of reception states, the **P-Queue** place contains only empty locations.

(i) The transition **T-Input** is crossable and generates a data packet in the first location of a queue.
(ii) The transition **T-Desynchro** is always crossable. We cross if for the first location in the preceding queue. This location then becomes desynchronized (as well as the location associated with it in the other queue).
(iii) The transition **T-Loss** can always be crossed and the data packet contained in this desynchronized location is lost. The location becomes empty.
(iv) The transition **T-Resynchro** is crossable; it generates a reinitialization packet in each location of the desynchronized pair.
(v) The transition **T-Transit** then allows the reinitialization packet to travel through the network, to the receiving end.
(vi) The transition **T-Output** then becomes crossable.

5.2.2. *Proof of properties* MP1 *and* MP2

Properties **MP1** and **MP2** concern sequentiality and the absence of packet duplication.

To prove these properties, we shall use the following theorem:

Theorem T1: the model does not duplicate packets and maintains their sequence.

Consider two distinct locations, on the same path and containing data packets:

1. *The packets are different.*
2. *The packet closer to the receiver (i.e. contained in the location whose number is greater) is the packet that was transmitted earlier (the packet number is lower).*

$\forall M \in \mathbf{A}$,

$\forall$ path $\in$ {Path1(M(**P-QUEUE**)), Path2 (M(**P-QUEUE**))}

$\forall u_i \in$ Path; $\forall u_k \in$ {Path u_i}

such that packettype (u_i) = packettype (u_k) = **DATA**
if locno(u_i) > locno(u_k)
then datano(u_i) < datano(u_k)

We prove this theorem in two stages: we start by showing that it is possible to restrict the model and to assimulate it to a simple FIFO queue, which gives us the properties we require. We then show that these properties are conserved in desynchronization management.

Stage 1: Assimilation of the model to that of a FIFO queue.

In transitions **T-Input, T-Transit** and **T-Output**, the components ⟨**state**⟩ and ⟨**direction**⟩ are not referenced in the triggering conditions, nor assigned in the output arc valuations. We can therefore map the tuplets of the model, restricted to these transitions, onto the tuplet ⟨**no, info**⟩. This therefore gives us the traditional model of a FIFO queue without loss [Berthelot 81]. It has been shown that this model has the properties of sequentiality (no desequencing of information carried) and non-duplication. Our model, restricted to these transitions, therefore also has these properties.

Stage 2: Conservation of properties in desynchronization management.

The properties of sequentiality and non-duplication are expressed by the components ⟨**no**⟩ and ⟨**info**⟩ of the tuplets.

(i) The transition **T-Desynchro** only modifies the ⟨**state**⟩ component of the tuplets; the other components are therefore not affected and the properties are conserved.
(ii) The transition **T-Loss** deletes the contents of the packet but does not modify the location number. The packet therefore becomes empty and no longer involved in the theorem.
(iii) The transition **T-Resynchro** replaces the information contained by a reinitialization packet without modifying the location number. Here again, the packet is no longer involved in the theorem.

5.2.3. *Proof of property* MP3

Property **MP3** concerns handling desynchronization. To prove it, we use theorem **T2**.

Theorem T2: Any desynchronization is correctly detected.

Consider a location containing a data packet.

If there is a location on the same path which:

(a) *is closer to the transmitter,* and
(b) *contains a data packet whose number is not consecutive (a loss has taken place between the two packets)*

then:

(i) *either there exists a reinitialization packet in one of the locations of lower number (closer to the transmitter),* or
(ii) *one of the lower number locations is desynchronized.*

$\forall M \in \mathbf{A}$,
$\forall$Path $\in$ {Path1(M(**P-Queue**)), Path2(M(**P-Queue**))}
$\forall u_i \in$ Path; such that packettype(u_i) = **DATA**

if $\exists u_k \in \{\text{Path} - u_i\}$ such that next $(u_i, \text{Path}) = u_k$ and packettype $(u_n) = \textbf{DATA}$
and datanumber$(u_i) >$ datanumber$(u_k) + 1$
then:

- *T21*: $\exists u_n \in \{\text{Path} - u_i\}$ such that:
  locno$(u_n) <$ locno(u_i) and packettype$(u_n) = \textbf{RE} - \textbf{INIT}$
- *T22*: $\exists u_n \in \{\text{Path} - u_i\}$ such that:
  locno$(u_n) <$ locno(u_i) and state$(u_n) = \textbf{DESYNCHRO}$

To prove that this theorem is an invariant of the model, we prove that it is satisfied in the initial state M_0 (which is trivial), and then that any transition crossing conserves the theorem.

For our demonstration, we need to introduce the following lemma:

Lemma L1: Any desynchronization affecting the last packet transmitted is detected.

Consider the packet closest to the transmitter; if it has not just been transmitted (its number does not correspond to the value of the transmitter counter decremented by 1) then:

(i) *either there is a reinitialization packet between it and the transmitter*, or
(ii) *there is a location between it and the transmitter which is the desynchronization state.*

$\forall M \in \mathbf{A}$,

$\forall$ Path $\in$ {Path1(M(**P-Queue**)), Path2(M(**P-Queue**))}

if $\exists u_i \in$ Path; such that last (Path) $= u_i$ and then datanumber$(u_i) <$ Counter(Path) $- 1$

let.*L11*: $\exists u_n \in \{\text{Path} - u_i\}$ such that:

locno$(u_n) <$ locno(u_i) and packettype$(u_n) = \textbf{RE} - \textbf{INIT}$

let *L12*: $\exists u_n \in \{\text{Path} - u_i\}$ such that:

locno$(u_n) <$ locno(u_i) and state$(u_n) = \textbf{DESYNCHRO}$

To prove the theorem, we study the behaviour of each transition:

1. Before crossing transition **T-Input**, on either path, there exists a last packet transmitted. According to lemma L1, any desynchronization concerning it will be detected. In the event of desynchronization, a location between it and the transmitter contains a reinitialization packet or is in the desynchronized state.
   The transition **T-Input** generates a packet for which the data corresponds to the value of the transmission counter. After crossing, theorem *T2* is therefore satisfied. The packet consequently becomes the final packet transmitted, its desynchronization is detected and we return to the situation above.
   Transition **T-Transit** causes packets to advance by transferring them into empty locations. Given that:
   (i) **T-Transit** modifies neither the information contained in the packets that are moving, nor the state of location;
   (ii) Empty locations are not taken into account by theorem *T2;*
   (iii) Every location number identifies its carrier tuplet uniquely.
   We can conclude that the transition **T-Transit** conserves theorem *T2*.
2. Transitions **T-Output** and **T-Loss** deletes packets by exchanging them for empty locations with the same number. As empty locations are not taken into account by theorem *T2*, it remains true.
3. Crossing of transition **T-Desynchro** takes a location from each transmission path to the desynchronized state. Consequently, this location satisfies, by construction, property *T22* of theorem *T2*.
4. Crossing of transition **T-Resynchro** replaces a location in the desynchronized state on each transition path, by a synchronized location containing a reinitialization packet.
   Before triggering the transition, property *T22* and, after crossing, property *T21* are satisfied.

Theorem *T2* is therefore satisfied for each transition in the model.

5.3. Functional validation

The functional validation of the model involves verifying that the properties of the model imply those of the service being modelled. Clearly, the complexity of this approach depends on the quality of the properties we have been able to derive from a model. In our example, functional validation is easy for the first three properties of the service, as we determined the properties of the model as a function of them.

Consequently, we can demonstrate trivially the following correspondences:

1. Model property **MPO** (liveliness of the model) corresponds directly to service property **SPO** (no irrecoverable blockage).
2. Model property **MP1** (conservation of sequentiality of the packets) implies service property **SP1** (no desequencing of packets).
3. Model property **MP2** (no duplication of packets) corresponds precisely to service property **SP2.**

The proof of correspondence between property **MP3** (handling of desynchronizations) and service property **SP3** requires the use of additional lemmas.

Lemma L2: Any loss of a packet is determined by the presence of a desynchronized location;

$\forall M \in \mathbf{A}$, if M(**T-Loss**) then $\exists u_i \in M$(**P-Queue**)
such that state$(u_i) = \textbf{DESYNCHRO}$

Lemma L3: Any resynchronization stage is concretely expressed by the transmission, on each transmission path, of a resynchronization packet.

$\forall M \in \mathbf{A}$,

if M(**T-Resynchro**) then $\exists u_i \in$ Path1(M(**P-Queue**))
such that packettype$(u_1) = \textbf{RE} - \textbf{INIT}$
and $\exists u_2 \in$ Path2(M(**P-Queue**))
such that packettype$(u_2) = \textbf{RE} - \textbf{INIT}$

Property **SP3** characterizes the behaviour of a network service in the presence of desynchronizations. We can prove, using lemmas *L2* and *L3* and theorems *T1* and *T2* that the model conforms to its specification:

1. From lemma *L2*, the transition **T-Loss** is only crossable if one of the locations is desynchronized.
2. According to lemma *L3*, the resynchronization stage (**T-Resynchro** transition) inserts a packet in each transmission path.
3. Theorem *T2* states that these insertions are made after every desynchronization.
4. The order of packets is conserved until they are delivered to the receiving end, and this takes place without duplication (theorem *T1*).

All these points allow us to establish that property **SP3** is satisfied, starting from the properties of the model.

6. Conclusion

In this article we have proposed a model for resynchronization services in the data transfer phase of the network layer. The model obtained is minimal as it brings together the required properties in a very reduced form. It is functionally equivalent to a protocol model describing all the internal mechanisms.

Our model can then be integrated with the model for the transport layer. Assertions relating to the network layer model are conserved in the transport layer model. These assertions can therefore be used for the demonstration of transport layer properties.

The functional validation approach for our model can be generalized to the characterization of any other hierarchical execution medium. The joint development of specification and validation models will make it possible in the near future to make tools available for specification, modelling and validation in a rigorous and easy way.

REFERENCES

[Ayache 85] J. M. AYACHE, J. P. COURTIAT, M. DIAZ, G. JUANOLE: *Utilisation des reseaux de petri pour la modelisation et la validation de protocoles (Use of Petri nets for modelling and validating protcols)*; TSI, **4**(1) January–February, 1985 (French edition).

[Azema 85] P. AZEMA: *Protocol analysis by using Petri nets*; IFIP 1985—Workshop on Protocol Specification Testing and Verification, 1985.

[Berthelot 81] G. BERTHELOT, R. TERRAT: *Petri nets theory for correctness of protocols*; IEEE Transactions on Communications, **COM 30** (12), 1981.

[Berthelot 81] G. BERTHELOT: *Transformation et analyse de R.d.P: application aux protocoles (Transformation and analysis of Petri nets: application to protocols)*; Higher doctorate, University of Paris VI, June, 1983.

[Brams 82] G. W. BRAMS: *Reseau de petri: theorie et pratique; (Petri nets: theory and practice)*; Vols. I and II, Masson 1982.

[Cousin 85] B. COUSIN, P. ESTRAILLIER: *Validation fonctionnelle de reseaux et predicats: Application au modele d'un protocole de communications (Functional validation of networks and predicates: application to a model for a communications protocol)*; MASI publication, Paris, 1985.

[Cousin 87] B. COUSIN: *Modelisation et Validation de systemes structures en couches (Modelling and validation of systems structured in layers)*; Doctoral Thesis, University of Paris VI, April 1987.

[Estraillier 86] P. ESTRAILLIER: *Conception de protocoles d'interconnexion robustes, (Design of robust interconnection protocols)*; Doctoral Thesis, University of Paris VI, 1986.

[Genrich 79] H. J. GENRICH, K. LAUTENBACH: *The analysis of distributed systems by means of predicate/Transitions nets*; Lectures notes in Computer Sciences No 70, Springer-Verlag, 1979.

[ISO-7498 83] ISO STANDARD: *Basic reference model on open systems interconnection*; ISO/Dis 7498, 1983.

[ISO-8072 84] ISO STANDARD: *OSI—Transport service definitions*; ISO/Dis 8072, 1984.

[ISO-8073 84] ISO STANDARD: *OSI—Transport protocol specification*; ISO-Dis 8073, 1984.

[ISO-8348 84] ISO STANDARD: *OSI—Network service definition*; ISO/Dis 8348, 1984.

[Transpac 79] TRANSPAC: *Transpace caracteristiques techniques d'utilisation des services—STUR, (Transpac technical characteristics for use of services—STUR)*; 1979.

[Zimmermann 80] M. ZIMMERMANN: *OSI reference model—the ISO model of architecture for open systems interconnection*; IEEE Transactions on Communications, **COM 28**, 1980.