A Protocol Specification Language with a High-level Petri Net Semantics

Belhassen Zouari[a], Serge Haddad[a] and Mohamed Taghelit[ab]

[a]Université Pierre et Marie Curie, MASI UA-CNRS
4, Place Jussieu, 75252 Paris Cedex 05, France
[b]Université de Tours, Faculté des Sciences et Techniques
Parc de Grandmont, 37200 Tours, France

Abstract

This paper deals with two important aspects of communication protocols namely *specification* and *verification.* We present a new variant of the Formal Description Technique Estelle called SSL which has the semantics of a High-level Petri net model. Such a semantics enables to apply efficient proof methods in order to automatically verify communication protocol properties. SSL is mainly characterized by a *total genericity* concept which misses in Estelle. We show that this specification feature has a direct effect on the efficiency of the verification methods, For instance, the SSL genericity feature implies a *symmetrical behaviour* of the corresponding Petri net. These *behaviour symmetries* allow the building of *reduced state graphs* called symbolic graphs. Moreover, SSL introduces some additional features of communication protocols as an object-based approach and high-level communication primitives. The major interest of SSL is to combine the advantages of a good specification language such Estelle with the analysis power of Petri nets.

Keyword Codes: C.2.2 ; C.2.4 ; I.6.4
Keywords: Computer-Communication Networks, Network Protocols, Distributed Systems; Simulation and Modeling, Model Validation and Analysis.

1. INTRODUCTION

The vital need for formalized specifications of distributed systems in general, and communication protocols, in particular, has led to the definition of Formal Description Techniques (FDT) within ISO (International Organization for Standardization) and CCITT. Three important FDTs have been arisen in the last years: SDL [1], Estelle [2] and Lotos [3]. The standard framework in which they have been defined has favoured the development of verification tools associated with these FDTs [4][5].

The main goals of defining formal techniques for specifying protocols are to provide languages that are [6]

1) expressive enough to represent all the *essential features* of a protocol in a clear and concise manner,
2) precise enough to *allow formal analysis* techniques to verify that the protocol as specified is free from logical errors.

The major difficulty is to meet *both* these goals with one formalism only. Indeed, formal verification approaches are based on simple and mathematically tractable models while specification approaches use rich and concise models in order to satisfy protocol designers requirements.

In this paper, we present a protocol specification model, called SSL (Symmetrical Specification Language), which has the feature to be supported by a High-level Petri net (HLPN) semantics. SSL is a good compromise between specification and verification objectives.

At the specification level, SSL is viewed as a dialect of Estelle [2, 7] enhanced by a generic and an object-based approach and by some additional protocol features (broadcast primitive, unreliable channels, ...). The SSL verification power lies in the fact that it has a Petri net semantics. Indeed, SSL specification can be translated automatically into HLPN from which one may apply the related verification methods.

The translation of a specification formalism into a verification one is getting an approach more and more adopted in numerous verification tools (e.g. [8, 9]). The originality of our approach is to show that a *well-structured* specification has a direct impact on *efficient verification capabilities.* In particular, concepts as *genericity and object-oriented* are not only sound qualities of a specification language but also relevant to the verification stage so as to build *reduced state graphs.*

In section 1, we discuss the Estelle limitations that led us to the definition of SSL as an enhancement of Estelle. Section 2 presents the main concepts of the SSL language and shows its capabilities in describing protocol features. In section 3, we present the HLPN model, called Well-formed Nets (WN), that makes up the semantics of SSL. We show that this model has a high structural and behavioural analysis power and particularly allows the building of reduced state graphs called *symbolic graphs.* Through an example, we show how an SSL specification can be translated into WN. In Section 4, we provide a partial SSL description of the Lamport's exclusion problem.

2. MOTIVATION

As Estelle is one of the FDTs that is becoming mature and accepted in industries, many Estelle support tools have been developed for use in the different stages of the protocol development cycle [4]. The success of Estelle may be explained by:

- the multiple concepts considered by the language (generic entities, process hierarchy, FIFO queues, time constraints, dynamicity, synchronous/asynchronous parallelism, ...) found in a broad range of types of protocols.
- its procedure oriented style. Indeed, a transition system and the *programming language Pascal* are used for expressing the dynamic behaviour.
- its "International Standard" status which favours its acceptation and spreading in the protocol users community.

Most of the Estelle verification tools are based on simulation approaches because it is procedure (and therefore implementation) oriented and is more similar to the programming languages [5]. Therefore, it is difficult to develop methods and tools which *allow formal verification* from Estelle specifications. Here, we outline the limitations of Estelle that bring about such difficulties and we show how to overcome them. Hence, SSL may be regarded as a variant of Estelle which introduces some restrictions and enhancements in order to make formal verification possible. One of our aims is to exploit the analysis power of HLPNs to verify communication protocols. HLPN models allow now the use of efficient methods in order to verify structural (invariant computation,...) and behavioural properties (reachability analysis, ...) of concurrent systems. Unlike the classical methods, we use *generic proof methods* [10][11] that avoid the "unfolding" of a HLPN which leads to a much lesser complexity. For instance, the building of reduced reachability graphs from a HLPN description has been studied in [12] [10]. These methods are based on an *equivalence relation* that exploits the behaviour symmetries in HLPNs. With the help of this relation, a node of the reachability graph is represented by an equivalence class. In [10], this graph is called symbolic reachability graph. Its main advantage is to preserve verification capabilities similar to that of the ordinary reachability graph.

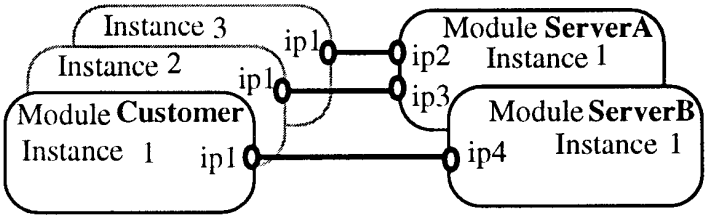
In general, communication protocols describe symmetrical systems. Indeed, the creation of many process instances from a *generic description* means that these instances have a symmetrical behaviour. Thus, processses with a symmetrical behaviour do not need to be *statically distinguished* in the verification process. This non-distinction is the basis idea to obtain a reduced size of the state graphs.

As a specification is made of entities and links, the genericity concept has two aspects: *architectural* and *behavioural. The architecture genericity means that all the instances of a given*

generic entity are connected similarly over the whole system. Such a genericity may be obtained if one defines generic links between generic entities. *The behaviour genericity implies that all the instances of a generic entity have the same behaviour description* (e.g. an automaton). Unfortunately, the genericity in Estelle is only *restricted.* Indeed, only the behaviour genericity is adopted in Estelle *since the comnunication links are defined between process instances and not between generic entities.* The following figure shows a graphical example of a system architecture specified in Estelle where the instances of a *generic process* (called *module)* are not similarly connected.

Figure 1.1: A restricted generic specification

In the opposite figure, we have three instances of the module 'Customer', an instance of the module 'ServerA' and an instance of the module 'ServerB.



Each module has a behaviour description represented by an extended automaton. Thus, all the 'Customer' instances have the same behaviour description. Nevertheless, the Customer instances are *differently* connected to the other system components. Indeed, Customer instances '2' and '3' are connected to the 'ServerA' instance while the Customer instance '1' is connected to the 'ServerB' instance. Here is a typical example of a restricted generic specification where the *architecture genericity* misses.

In [13, 14], it is demonstrated that the restricted Estelle genericity is the main reason which leads to a non-symmetrical behaviour of the specified system. Moreover, Estelle specifications allow to statically distinguish the process instances. Therefore, the major interest of the HLPN verification techniques as the symbolic reachability graph is lost. To tackle this problem, SSL provides a total genericity concept (i.e. both architecture and behaviour genericity) which allows us to exploit the optimized verification techniques of HLPN.

2. SYMMETRICAL SPECIFICATION LANGUAGE (SSL)

The previous motivations lead us to define a protocol specification model, called Symmetrical Specification Language (SSL). A SSL specification is made of two parts:

- an architecture description which defines the sequential components of the system, the types and the communication links, and
- a behaviour description which defines the behaviour of each component.

2.1. SSL concepts

Four important features distinguish SSL from formal description techniques like ESTELLE (which is strongly inspired from):

- *genericity* which is also applied to links,
- *Object-based approach* which makes it more natural, concise and rigorous,
- *high level communication primitives* which are a necessity for real requirements,
- *formal semantics* which allows formal verification techniques

The last point is discussed in the next section.

2.1.1. Genericity

SSL allows one to define not only generic communicating entities but also generic links. Instances (also called processes) of a given generic entity are created by defining a set of communicating entities called *group.* A group represents a set of potentially *symmetrical* processes. The architecture description is based on the definition of these groups. To define an architecture, *construction operators* are used. These operators are applied to groups only and not to their elements. Hence, the processes symmetry is preserved.

The system behavioural description consists in defining an extended state machine for each processes group. The state machine describes the behaviour of each process in terms of states and transitions. As such a *description* is *common to a set of processes)* (i.e. a group instances),

the *behaviour genericity* is met. Just as the architecture construction operators, the operators used in the behaviour description preserve the behaviour symmetry in the groups.

In SSL, only generic links are allowed since a link definition is achieved between *groups* and not directly between group instances. This ensures that all the instances of a group are connected *in a same manner* which precisely leads to the *architecture genericity* of SSL specifications. Note that a generic link represents a set of connections between group instances. This requires to specify the kind of connections involved by the definition of a generic link. This is achieved in the link declaration by means of a specific attribute which determines basic topologies such as multicast and ring.

From static point of view, the instances of a same group can not be distinguished (no static identities are associated with instances). As the specification is totally generic (both architecture and behaviour aspects), the behaviour of these instances within the system is symmetric. However, their progress (expressed by firing transitions in the associated automaton) may dynamically distinguish them. For instance, a non-deterministic occurrence of events (e.g. a message reception) may lead the processes to fire different transitions and so, to progress differently.

2.1.2. Object-based approach

The *process* and the *group* notions are two main features of the SSL language. A process is similar to an *object* and a group is similar to a *class*. This means that any group defines a type of the language. This type is not equivalent to an interval or an enumeration type because it forbids constants use. The fundamental principle of SSL is that a process only knows the processes it interacts with. *Hence, the processe, are never explicitly referenced.* When necessary, process identities may be sent in messages or memorized in variables. Initially, a process only knows its own identity stored in a predefined variable (usual approach of the object languages).

This makes SSL more natural, concise and rigorous. These object and class notions are already integrated to the SSL language and some other ones, related to object approach, such as inheritance and polymorphism are planned for the future versus.

2.1.3. High-level communication primitives

SSL includes a set of communication primitives which fit the requirements of real applications. For example, to reference a target entity in a message sending command, the following capabilities are offered:

- selection of any entity in a given group (non-deterministic choice),
- selection of the whole entities of a given group (used for broadcast),
- selection of the "successor" entity in case of ring architecture,
- selection of one (or more) entity previously memorized in a local variable.

These capabilities are powerful enough to take into account the most important requirements of real systems. This, while they preserve the natural, concise and rigorous characteristics of the language. Some other capabilities, relevant to communication features such as message lost rate are planned for the future versus of the language.

2.2. SSL Language

A specification in SSL is made of two parts: a first part that defines the *architecture* of *the system* and a second one which describes the *behaviour of its components.* In the following, we give a brief description of SSL through an example. This presentation must be regarded as a stress on the main capabilities of the language rather than an exhaustive description.

2.2.1. The Architecture Definition

The specification of the architecture defines an hyper-graph whose vertices are communicating entities and whose edges are communication links between entities. The construction of such a graph requires two stages:

- a declaration of groups, types and links,
- a sequence of construction operations.

Declaration part

The declaration part includes a list of group identifiers, type declarations and link declarations. A group identifier refers to a set of communicating entities that must have a similar behaviour. The cardinality of each group (i.e. the number of instances) is definitively determined just at the end of the construction part.

The following statement is a declaration of three groups :

```
GROUP manager, computer, terminal;
```

Initially, each group (manager, computer and terminal) contains one instance.

A link declaration holds the following information:

◊ a type: it defines the structure of the messages allowed on the link.

◊ a basic configuration: as a SSL link is generic, it corresponds to a set of identical link instances. Hence, two basic configurations are defined:

- SUCCESSOR: it allows to define ring topology on a set of processes.
- PRODUCT: it allows to define a multicast topology on a set of processes.

These basic configurations are *combined* together using construction operators so that any desired system topology may be defined (see next section).

◊ a discipline: two disciplines are available, "FIFO" to specify the sequencing of transmitted messages and "MAILBOX" to specify their non-sequencing.

```
TYPE type_msg = (req, ack, token);
```

This type declaration describes three messages to be sent. Structured messages may be defined.

```
LINK man_comp, term_comp : type_msg SORT PRODUCT DISCIPLINE MAILBOX
     on_ring:     type_msg SORT SUCCESSOR DISCIPLINE FIFO
```

The previous link declaration defines two links. The `man_comp` and `term_comp` links specify point to point connections between all the instances of two groups. It means that when one of these links is created, every existing instance of the given group is connected to all the existent instances of the other group at this same time. Similarly, the `on_ring` link specifies a ring connection between all the instances of the given group.

Construction Part

The construction part defines in *an incremental manner* the architecture of a system. This construction operates on a set of *structures.* The structure is a semantic notion which is not explicitly handled by the specifier. It is defined as a set of groups bound by links. Initially, each declared group represents an *elementary structure* which is made up of just the group. Initially, every declared group has a cardinality of one, which represents one instance. Starting from elementary structures, one may build more complex structures using two basic operators *(associate* and *duplicate)* until one obtains the desired architecture. Let us take the following architecture and show how we can obtain it by means of the construction operators.

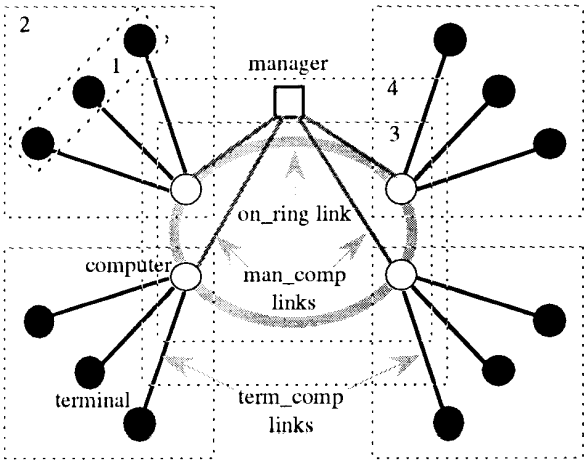


Figure 2.1: Example of architecture

The opposite architecture is made up of a manager, 4 computers and 12 terminals. The manager is connected point to point to 4 computers. Each computer is connected point to point to 3 terminals. Computers communicate according to a ring based protocol.

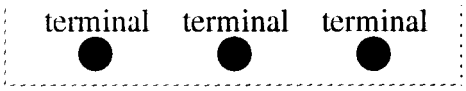
In SSL, the construction of architecture is achieved in four steps illustrated by dotted rectangles in Figure 2.1.

First, three groups are defined by the next statement:

```
GROUP manager, terminal, computer;
```

One of the two construction operators is the duplication operator. Intuitively, *duplication* is the manner to create instances of both processes and connections. Hence, a duplication takes a structure and one integer value as inputs and produces a set of identical structures which cardinality is the input value. The syntax and the semantics of this operation is illustrated by the following example.

```
terminal[3];
```

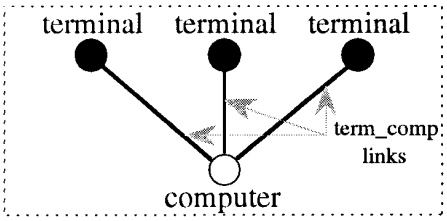


The structure above (which is an elementary one) is produced by the previous operation. It corresponds to the first step of the architecture of Figure 2.1. The terminal group is duplicated three times.

The second construction operator is the association one. Intuitively, *association* allows to create a link between two groups (which may be or not be the same). Association takes two group identifiers and one link identifier as inputs. An association defines a set of connections between the instances of two groups (eventually of one group) according to the characteristics of the link. Hence, it produces a new structure from the ones including the group identifiers.

```
ASSOCIATE (terminal, computer, term_comp);
```

The opposite structure is produced by the previous operation. The `term_comp` link connection type specifies that each instance of the `terminal` group is connected to all instances of the `computer` group. At this stage, the `computer` group is made up of only one instance.



One can abbreviate an association followed by a duplication with the following syntax.

```
ASSOCIATE (terminal, computer, term_comp) [4];
```

This produces a structure made up of four instancess of the structure shown above. It corresponds to the second step of the architecture construction related to Figure 2.1.

In the example, the computers communicate according to a ring based protocol. In this case, we use the following operation to define a link on the `computer` group:

```
ASSOCIATE (computer, computer, on_ring);
```

The on_ring link specifies that all the instances of the computer group are connected through a communication ring. It corresponds to the third step of the architecture construction in Figure 2.1.

In the example, the manager is connected to all the computers. This is achieved by the following operation:

```
ASSOCIATE (manager, computer, man_comp);
```

This last operation, corresponding to the fourth step of the architecture construction related to Figure 2.1, produces the final structure.

Let us give now the complete architecture definition of the example.

```
SPECIFICATION ring_net ;
GROUP
     manager, computer, terminal;
TYPE
     type_msg = (req, ack, token);
LINK
     term_comp, man_comp: type_msg SORT PRODUCT DISCIPLINE MAILBOX
     on_ring:      type_msg SORT SUCCESSOR DISCIPLINE FIFO
```

```
ARCHITECTURE
     terminal[3] ;
     ASSOCIATE (terminal, computer, term_comp) [4];
     ASSOCIATE (computer, computer, on_ring);
     ASSOCIATE (manager, computer, man_comp);
```

Program part I : Architecture description

2.2.2. The Behaviour Definition

The behaviour of a system is related to the behaviour of its components which are processes. The behaviour of a group processes is described by an extended automaton. Indeed, any transition of the automaton involves a *guard* part and an *action* part. The guard part consists of conditions that must be satisfied in order to fire the transition. These conditions are either related to the reception of a message (WHEN clause), or to the evaluation of a Boolean expression (PROVIDED clause) or to a timer (DELAY clause). The action part may include sending primitives and/or internal treatment. The internal treatment is a sequence of elementary operations commonly used in programming languages (as assignment statement and arithmetic operations). These operations handle variables *local* to a process.

The types associated with variables may be either:

. *simple* such as enumeration, interval and group, or

. *constructors* such as arrays, records and group sets.

Note that the 'group' and the 'group set' types are specific to SSL. A 'group' variable allows to memorize an identity of a particular process of a group while a 'group set' variable allows to memorize a set of process identities of a given group. For a process, a typical use of a group variable is to memorize a process identity during a message exchange.

Let us now consider the behavioural aspect of the protocol seen below. In order to make the complete study of the example (specification, translation and verification aspects) possible in this paper, we have simplified the protocol. Therefore, some important SSL features may not appear here but can be found in [13].

The system consists in regularly testing the state of the terminals connected to the network. Initially, the manager initializes the network by providing a 'token' to anyone of the computer processes. Then, the computer owning the token tests the state of its terminals. It broadcasts a request message and waits for the acknowledgments. Afterwards, the computer frees the token and sends it to its successor on the ring. Terminals have only one action namely acknowledging the request messages.

The SSL specification of this protocol is the following:

We assume that a terminal has one state: `t_idle`. On reception of a request message (`req`) from the computer, it sends an acknowledgment message (`ack`) and returns to its `t_idle` state. The terminal behaviour may de described as follows:

```
BEHAVIOUR terminal;
INIT   t_idle;                                                    /* 1 */

TRANS t_send FROM t_idle TO t_idle WHEN req ON term_comp           /* 2 */
       BEGIN OUTPUT ack ON term_comp END;
```

Program part II : Terminal behaviour description

(1) Initially, a terminal is set to its single `t_idle` state.

(2) The `t_send` transition is enabled if the terminal is in its `t_idle` state and a `req` message is available on the `term_comp` link. The `t_send` transition firing implies the output of the `ack` messsage on the `term_comp` link. This means that the addressed entity is the computer to which the sender terminal is connected. Then, the terminal returns to the same state.

We assume that a computer has two states: c_idle and c_wait. The computer behaviour may be described by the following text:

```
BEHAVIOUR computer;
VAR   nb: INTEGER SET TO 0;                                          /* 3 */
INIT c_idle;                                                         /* 4 */

TRANS c_rec_m FROM c_idle TO c_wait                                  /* 5 */
      WHEN token ON man_comp
      BEGIN OUTPUT req ON term_comp TO ALL END;

TRANS c_rec_c FROM c_idle TO c_wait                                  /* 6 */
      WHEN token ON on_ring
      BEGIN OUTPUT req ON term_comp TO ALL END;

TRANS c_rec_t FROM c_wait TO c_wait                                  /* 7 */
      WHEN ack ON term_comp PROVIDED (nb < 2)
      BEGIN nb := nb + 1; END;

TRANS c_send FROM c_wait TO c_idle                                   /* 8 */
      WHEN ack ON term_comp PROVIDED (nb = 2)
      BEGIN OUTPUT token ON on_ring; nb := 0; END;
```

Program part III: Computer behaviour description

(3) nb is declared as an integer variable. It is used by a computer to memorize the number of ack messages it has already received from the terminals connected to it.

(4) Initially, a terminal is set to its c_idle state.

(5) A computer fires this transition if it is in its c_idle state and if the token is available on the man_comp link. In this case, the token is sent by the manager and corresponds to the initialization of the network. So, this transition is fired only once by one computer only. This computer is arbitrary chosen by the manager.
When the computer fires this transition, it broadcasts on the term_comp link a req message to all the terminals connected to it. Then, it enters its c_wait state.

(6) The c_rec_c transition is similar to the previous transition. The difference is that the computer receives the token from its predecessor (computer) and not from the manager.

(7) The c_rec_t transition is enabled if the computer is in its c_wait state, an ack message is available on the term_comp link and the value of its nb variable is less than 2. This means that the ack message is not the last one it is waiting for. Then, it increments its nb variable and returns to the same state.

(8) A computer fires the c_send transition when it receives the third ack message it is waiting for. It sends then the token to its successor, sets to zero its nb variable and enters its c_idle state.

The manager behaviour may be described as following:

```
BEHAVIOUR manager;
INIT m_idle;                                                         /* 9 */

TRANS m_init FROM m_idle TO m_death                                 /* 10 */
      BEGIN OUTPUT token ON man_comp TO ANY END;
```

Program part IV : Manager behaviour description

(9) Initially, the manager is set to its m_idle state.

(10) The m_init transition is the first one enabled in the system. Its firing initializes the network by sending on the `man_comp` link a token to anyone of the computers. The target computer is chosen arbitrary due to the SSL parameter '`TO ANY`'.

3. SSL SEMANTICS: AN INTRODUCTION TO WELL-FORMED NETS

We present in this part the High-level Petri net model (the well-formed nets) which provides a semantics for SSL. After an informal introduction, we describe the different verification tools available for this model justifying the interest of this semantics in addition to its simplicity. At last we give the semantics rules which transform a SSL program into a well-formed net.

3.1. Definition of a well-formed net

3.1.1. Informal Presentation

The well-formed nets model [10] is an extension of the Petri nets model. The aim of this extension is to overcome the inability of Petri nets to provide a concise description of real systems. We will not give a formal definition of this model that would be out of the scope of this paper but we will rather point out the main features of this extension:

- In order to keep more information in places of the nets, *the tokens are coloured by an information.* The domain of this information is specific to each place and is called the coloured domain of the place. For instance the coloured domain of a place modelling a process state is the set of processes (i.e. tokens) showing by this way which processes are in this state.
- In order to avoid duplication of transitions' modelling the same actions but applied on different objects, *the firing of a transition is coloured by an instance.* The domain of these instances is specific to each transition and is called the coloured domain of the transition. The coloured domain of a transition modelling the catch of a resource by a process is the set of couples <process, resource> showing which process takes which resource.
- As it may be noticed on the examples, the colour domains are structured. In well-formed nets, every domain is a Cartesian product of object classes *where the classes are the primitive domains.* An object class may be divided into static subclasses. The meaning of this partition is the following: objects with the same structure are grouped in the same class and objects with the same potential behaviour are grouped in the same static subclass. For instance the processes could be grouped in a class split in two subclasses: interactive processes and batch processes.
- The valuation of an arc between a place and a transition can no longer be an integer. Instead, it must be *a colour function sspecifying the coloured tokens consumed or produced by each coloured instance of the transition.* For instance if the catch of a resource by a process requires an idle process, then the function valuating the arc between the "catch" transition and the "idle" place maps a couple <process,resource> on its first item.
- Once again one can notice that the colour functions are structured. In well-formed nets, every colour function is built by standard operations (linear combination, composition,…) on basic functions *where these basic functions are the primitives assssociated with the object classes.* For instance the identity function is a 'trivial' primitive of an object class.

3.1.2. A database management modelling

We are now going to introduce all example in order to illustrate the well-formed nets model. A database has multiple copies, each one owned by a site. In order to modify the database, a site must take a grant (we will ignore in our modelling the access mechanism to this grant). Once the modification is done, the site sends update messages to all the other sites. On reception of such a message, each site updates its own database and sends back an acknowledgment. After having received all the acknowledgments, the owner of the grant frees it.

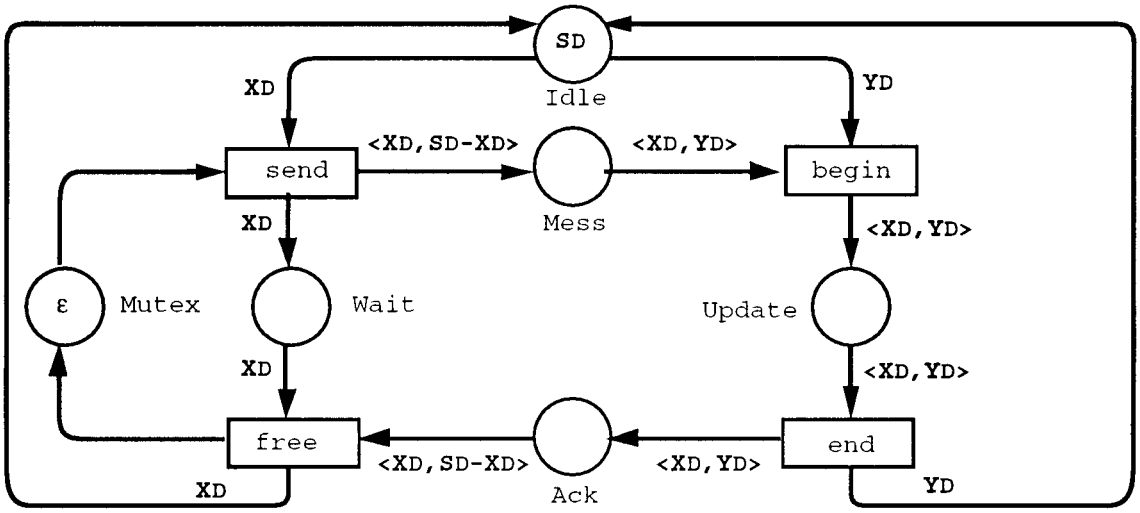


Figure 3.1 : A well-formed net model of a database management with multiple copies

We give the skeleton (graph structure) of the net and we describe the executed protocol in the case of a database modification. An *Idle* site takes the grant for *Mut*ual *Ex*clusion, modifies its own database and *sends Mess*ages to other processes; then it *Waits.* On reception of a *Mess*age, an *Idle* site *begins* its *Update.* At the *end* of its *Update,* it sends back *Ack*nowledgment and becomes again *Idle.* The *Waiting* site frees the grant for *Mut*ual *Ex*clusion when it receives all the *Ac*knowledgments and becomes again *Idle.*

Let us take a look on the colour structure of the net:

- there is only one object class: the set of sites called D. This domain has two primitives: the identity functions denoted X_D or Y_D (X, Y for free variables) and the diffusion function which selects all the sites denoted S_D (S for sum).
- there are three kinds of colour domains: the object class D which is for instance the colour domain of *Idle* or *send,* the product $D \times D$ which is the colour domain of *Mess,* as a message includes the identity of the sender and the receiver and the neutral domain $\{\varepsilon\}$ which is the domain of *Mutex* as no information is necessary for the token in this place.
- we will just present the more complex function $<X_D , S_D - X_D >$ of the net which appears on the arc from *send* to *Mess.* When a site c fires the transition, one must have all the tokens <c,c'> for $c' \neq c$. How is it obtained ?

  (a) the constant function S_D always produces $\sum c'$ whatever the item applied on it is.

  (b) the function X_D is the identity and produces the token c.

  (c) thus the function $S_D - X_D$ produces the tokens $\sum c'$ for $c' \neq c$.

  (d) at last the function produces the symbolic token $<c,\sum c'>$ for $c' \neq c$ which is equivalent to $\sum <c, c'>$ for $c' \neq c$.

- the initial marking represents one neutral (ε) token in *Mutex* and one token per site in *Idle* (S_D).

3.1.3. Discussion

The well-formed nets model is one of the high-level formalisms extended from Petri nets. The advantages of this model are fourfold:

- **Structure:** Colour domains and functions are structured and syntactically well defined.Thus researchers have been able to extend the analysis methods on Petri nets based on graph structure: reductions of net [15] and deadlocks research [16].

. **Algebra:** Colour functions are obtained from elementary functions by algebraic operations on functions such as external product by a scalar, sum, noetherian product,... Thus net functions are items of a finite generated algebra and algebraic methods on Petri nets can be extended like the flows computation [11].

. **Genericity:** As object classses are parametrized and primitives may be applied to any class, a well-formed net defines a family of models. Thus, we have true genericity which is also exploited at the specification level of the language SSL [14].

. **Symmetry:** The objects in a subclass have the same potential behaviour. This is why researchers have exploited this behaviour symmetry in order to build a reduced reachability graph where a node is a set of states and where an edge is a set of transition firings [10].

3.2. Analysis methods

The methods available for well-formed nets can be classified in three categories: algebraic methods which take into account the linear equations defining the change of states, structural methods which consider the graph structure of the net to deduce some properties and the dynamic methods which use the reachability set (partly or totally). In the following, we present one example per category. Moreover, there exist other methods such as deadlocks computations which are sometimes structural and sometimes algebraic.

3.2.1. Flows Computation

An invariant is a property which is true for all the reachable states. Many models provide the way of verifying that a property is an invariant. Nevertheless in Petri nets and well-formed nets there exists a category of invariants called linear invariants for which a generative family can be computed.

The theoretical basis of these methodss is the state equation of nets which relates to the transformation of a state in another by a firing sequence and can be expressed by :

M' = M + W.**s** where M' is a marking obtained from M by the sequence s and where W is the incidence matrix and **s** is the occurrences vector of transition in s. Thus, linear invariants can be deduced from vectors X which fulfill X.W=0. Such vectors are called flows.

In Petri nets, the computation of a generative family of flows is easily done by a Gaussian elimination. In well-formed nets the computation is much harder as the items of the incidence matrix are no more integers but functions. However some important results have been obtained. We give now a generative family of flows for our example.

(1) State of a site:

$\mathbf{X_D}$.Idle + $\mathbf{X_D}$.Wait + Y_D.Update

(2) Either the grant is free, or a site iss waiting:

1.Wait + **1**.Mutex

(3) For a site c waiting, the tokens <c,c'> are distributed between places Mess, Update and Ack :

$-\langle \mathbf{X_D}, \mathbf{S_D}-\mathbf{X_D} \rangle$.Wait + $\langle \mathbf{X_D},\mathbf{Y_D} \rangle$.Mess + $\langle \mathbf{X_D},\mathbf{Y_D} \rangle$.Ack + $\langle \mathbf{X_D},\mathbf{Y_D} \rangle$.Update

A generative family of flows

3.2.2. Net reductions

A net reduction is given by application conditions and a transformation method such that if the initial net fulfills the conditions, then the reduced net has the same behaviour as the original net. The interest of the reductions lies in their broad application field and the simplicity of the verification and the transformation (structural examination and flows computation).

The reductions have been extended for well-formed nets following the principles below:

. the structural conditions have been preserved,

. only necessary functional conditions have been added where the conditions can be verified by syntactic analysis,

. transformations have been extended with standard operations on functions (composition, inverse,...)

The correctness of the well-formed net reductions has been proved by pointing out that they are equivalent to a sequence of ordinary reductions applied on an equivalent ordinary Petri net.

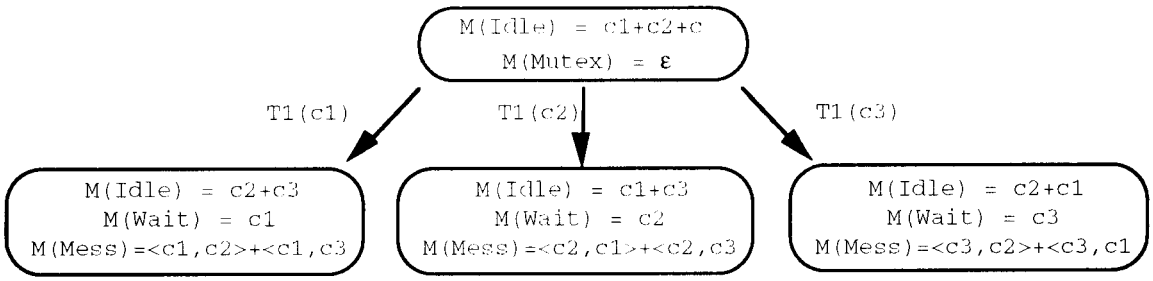
For instance, the database management net has been reduced to a single transition showing by this way liveness and safeness of this protocol.

3.2.3. Symbolic graph building

The key point of the building of a symbolic graph is the observation that numerous symmetries exist in well-formed nets and that these symmetries also exist in reachable markings. The principle of the method is to define an equivalence relation between objects of the same subclass and to extend it to different sets: colours of a domain, firing instances of a transition and markings of a place. Rather than building the ordinary reachability graph and testing the symmetry relation on the fly, the method proposed in [10] works directly with symbolic markings and symbolic firings which are representations of equivalence classes. The technique of representation of a symbolic marking is based on partitioning each static subclass of objects in virtual subsets called dynamic subclasses. A dynamic subclass is intended to group objects with the same state in the current marking. For any consistent instanciation of the dynamic subclasses by concrete subsets, we get an ordinary marking. The firing rule is then adapted to work with symbolic markings.

On this graph one can deduce by examination the main properties of the net behaviour (e.g. safeness, liveness, home state,...). Moreover in [17] it is shown how to extend the building to take into account the stochastic well-formed nets and to efficiently obtain the stationary probabilities of the states. Last but not the least, the symbolic representation is more significant for the modeller than the ordinary one.

In the database modelling net, one can initially fire the transition T1 and that for any instanciation (i.e. for any site). This firing represents a local modification and the sending of corresponding messages as shown below:



The same firing step is represented in the symbolic graph as shown below. Let us detail this subgraph.

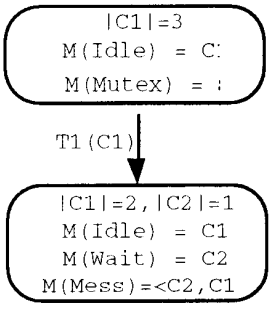


Figure 3.4 : A symbolic firing of the database management net

The first node is the initial symbolic marking. In this mar.king, all the sites are in the same state so the class of sites is partitioned in one dynamic subclass called C1 with a cardinality 3. The symbolic firing instanciation of Tl is made by this subclass. In the new marking, the class of sites is now partitioned in two dynamic subclasses: the waiting site and the sites with a message to handle. The interpretation of such a marking is natural for the modeller : one site has done a modification and the other ones have not begun their corresponding update.

3.4. Translation of SSL specifications into Well-formed Nets

In this section, we present the main principles in translating a SSL specification into a WN. This translation is achieved in many stages. First, we provide an informal description of every stage. Then, we show how the protocol previously presented is translated into a WN.

The translation of the architecture part may be divided into two stages:

- The translation of the *construction operations* in a SSL specification lead to the generation of *object classses* and to the computation of some *colour domains.*
  Intuitively, a *duplication* operation generates a *new object class* since this duplication corresponds to the creation of new instances (of processes and eventually of links). As a process is represented by a colour in the resulting WN, a colour domain is associated with each group. This domain is updated at every duplication involving a given group. Hence, the structure of the tokens (i.e. simple or n-uple tokens) representing the processes of a group as well as its cardinality are determined by the computed domain.
  This domain also corresponds to a group type.
  An *asssociation* operation defines the structure of messages allowed on the specified link and generates a *WN substructure* (places and eventually a marking) so as to store the transmitted messages. This subtructure depends on the characteristics of the link (FIFO, mailbox, successor, product).
- The declared *types* in SSL are translated into *object classes* in WNs. A scalar type (i.e. enumeration or interval) directly corresponds to an object class. As the structured types are made up of scalar types, their translation is just the computation of the corresponding colour domain.

The translation of the behavioural description is achieved in three stages:

- We generate a *WN subsstructure* corresponding to the skeleton of a given group *automaton.* The states of the automaton are translated into WN places which colour domain has been computed in the architecture translation stage. The transitions of the automaton are translated into "incomplete" WN transitions. In a further translation stage (translation of guards and action), we add arcs to these ""incomplete" transitions. The place corresponding to the initial state of a group is marked by the set of tokens representing the group instances.
- The translation of local variables is achieved by generating a place by simple or array variable. The record variables are considered as a set of simple variables, so their translation may lead to the generation of many places.
- The previous translation stages generate substructures which correspond either to the SSL automata, to links or to variables. This stage allows to compose all these substructures by translating the guards and the actions associated with SSL transitions. In fact, this translation generates WN predicates and synchronizing arcs that bind the substructures previously obtained.

Let us illustrate these translation stages by an example. The following figure shows the WN resulting from the translation of the protocol described in the previous sections. Many substructures can be viewed: three ones corresponding to the automata of the groups, three others corresponding to the three links and one corresponding to a local variable (i.e. 'nb'). The arcs binding these substructures correspond to either communication operations or to variable access.

The corresponding information iss associated with thiss net:

Object classes:

$C_c = \{c_1, c_2, c_3, c_4\}$ corresponds to the 'Computer' instances.

$C_t = \{t_1, t_2, t_3\}$ corresponds to the 'Terminal' instances connected to one computer.

$C_m = \{\varepsilon\}$ corresponds to the 'Manager' instances.

C_{msg} = {req, ack, token} corresponds to the declared type 'type_msg'.

$C_{integer}$ is a predefined type containing the values from 0 to 'maxint'.

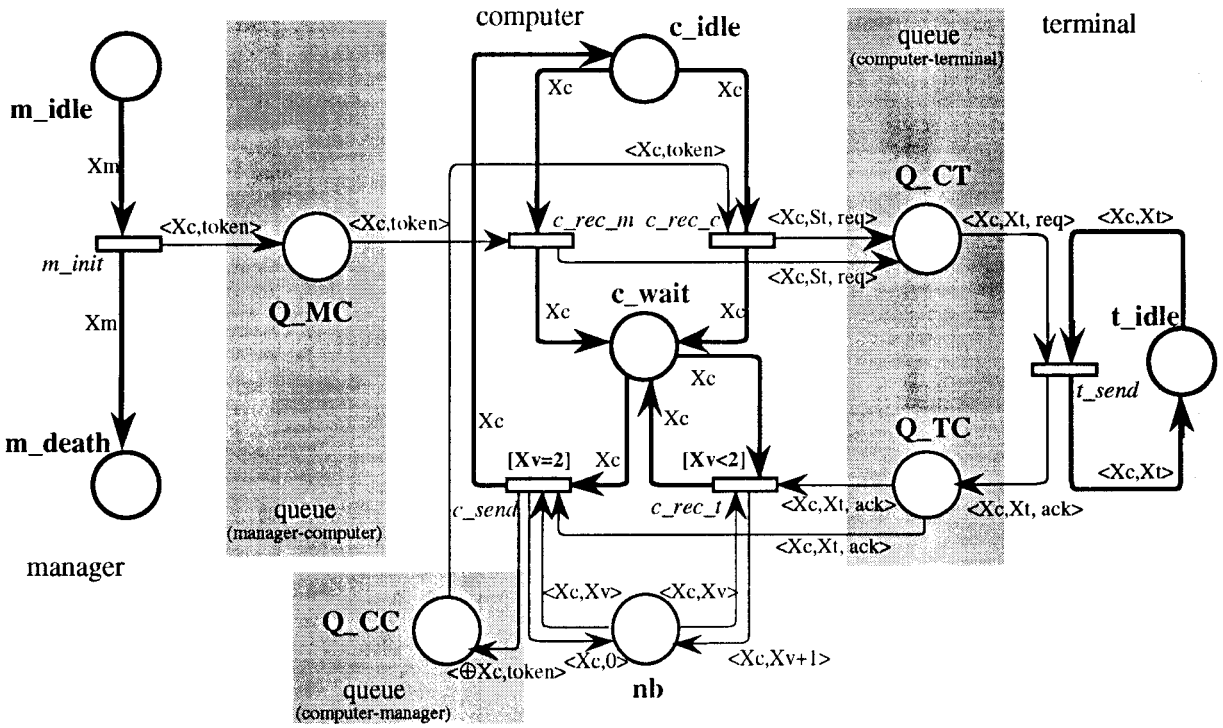


Figure 3.2 : the WN corresponding to the ring_net protocol

Colour domains

$D_{computer} = C_c$ is the colour domain associated to the computer group. It represents all its instances as well as the initial marking of the place 'c_idle'.

$D_{terminal} = C_c \times C_t$ is the colour domain associated to the terminal group and represents all its instance,s. So a terminal process is represented by a token made up of a computer identity and terminal one (i.e. $<c_i, t_j>$).

$D_{manager} = C_m$ is the colour domain associated to the manager group.

The places in boxes 'manager', 'computer' and 'terminal' (see Figure 3.2) have a colour domain equal to that associated with the corresponding group. For example, the colour domain of the place 't_idle' is equal to $D_{terminal}$. The colour domain associated with the link places is computed differently. It depends on the messsage type and both the sender and the receiver colour domains. For instance, the place Q_CT represents the queue of the link 'term_comp' in the direction computer towards terminal. The colour domain of this place is made up of the sender domain (C_c), the receiver domain ($C_c \times C_t$) and the message type domain (C_{msg}) which provides $C_c \times C_t \times C_{msg}$.

Colour functions

The following functions allow the selection of one object from the associated classes:

X_c for the class C_c; X_t for the classs C_t; X_m for the class C_m and X_V for the class $C_{integer}$;

S_t is a diffusion applied to the class C_t. In Figure 3.2, it represents the broadcast of the request message to the terminals connected to the sender.

$\oplus X_c$ allows the selection of the object *successsor* to the one selected by X_c (this function is allowed on *ordered* object classes only).

Initial marking

The following places are initially marked:

`m_idle` by the neutral colour (i.e. the 'ε' token)

`c_idle` by all the tokens defined by the computer domain (i.e. C_c)

`t_idle` by all the tokens defined by the terminal domain (i.e. $C_c \times C_t$)

`nb` by the initial value '0'

The complete presentation of the translation stages is formally defined in [13].

4. EXAMPLE

This section aims at bringing out the conciseness of the SSL language in describing real examples. From a mutual exclusion problem, we provide a partial SSL specification. The distributed algorithms of mutual exclusion are excellent illustrations of the genericity concept, since hosts arc connected among themselves and have to manage in a symmetrical manner (and fairly) their entry in the critical section. We briefly remind the Ricart and Agrawala's algorithm [18]. Each host has a counter initially set to *n-1* (where *n* is the hosts number) and, when this counter reaches the zero value, the host may enter the critical section. Initially, the host broadcasts a request to all the others hosts. Then, it decrements its counter on receipt of each acknowledgment message or each request message that is more "recent". The hosts which sent these requests are memorized and when the host exits its critical section it broadcasts an acknowledgment message to this subset of waiting hosts. Finally, when a host is idle, it sends an acknowledgment message on receipt of each request message.

Here is the specification of the mutual exclusion algorithm. This specification only describes a partial behaviour

```
SPECIFICATION Ricart_Agrawala;
GROUP
      processes;                              { a unique group: the processes}
TYPE                                          { one kind of message: clock vaalue amd nature}
      tp_req =(req, ack);
      tp_mess = RECORD c: INTEGER; r: tp_req END;
LINK                                          { meshed link preserving the messages order}
      complete: tp_mess SORT PRODUCT DISCIPLINE FIFO;


ARCHITECTURE
      processes [N];                                                    /* 1 */
      ASSOCIATE (processes, processes, complete);                       /* 2 */



BEHAVIOUR processes;                          {generic behaviour of the processes}
VAR
      m_temp      : tp_mess;                  { last received message}
      p_temp      : processes;                {process which sent the last received message}
      defer       : SET OF processes;         {set of processes to acknowledge}     /* 3 */
      clock       : INTEGER SET TO 1;         {clock of the process}
      old_clock   : INTEGER;                  {time of the last request}
      waiting     : INTEGER;                  {number of waiting proocesses}

INIT  idle;                                   {initial state}

TRANS request_section FROM idle TO wait
      BEGIN
         OUTPUT <clock, req> ON complete TO OTHERS;                     /* 4 */
         old_clock := clock;
         clock := clock + 1;
         defer := EMPTY;                                                /* 5 */
         waiting := N -1;
      END;
```

```
TRANS exit_section FROM critical TO idle
      BEGIN
          OUTPUT <clock, ack> ON complete TO defer;                    /* 6 */
          clock := clock + 1;
      END;

TRANS wait_recept FROM wait PRIORITY 1
      PROVIDED trait_recept
      BEGIN
          IF (m_temp.c > old_clock) THEN
          BEGIN
              waiting := waiting -1;
              IF m_temp.r = req THEN defer := defer + p_temp;          /* 5 */
          END;
          trait_recept := FALSE;
      END;
```

Comments

(1) The first primitive of the construction architecture is a *duplication.* This one leads to the definition of `N` processes within the '`processes`' group.

(2) The second primitive of the architecture construction is the *association* between groups by means of the generic link `complete`. This one allows each process to directly communicate with any other process.

(3) The defer variable contains a subset of processes identities. The processes group is a specific type which has the use of new type constructors such as SET OF.

(4) The broadcast of a message to the other processes is done by one statement since the *receiver* may be a subset of processes. Here, this subset is specified by means of the keyword OTHERS which represents all the other processes.

(5) A process is never explicitly pointed out. Thus, the defer variable is initially set with the keyword EMPTY and updated with the value of a process type variable. On receipt of a message, this variable is updated with the sender identity.

(6) The selective broadcast does not bring about any particular problem since the *receiver* is specified by a variable which determines a set of target processes.

S. CONCLUSION

In this paper, we have shown how genericity may be considered in studying protocols. This study has covered two essential aspects of a protocol life-cycle: specification and verification. We have presented the SSL language which is characterized by numerous protocol features that we can find in the FDT Estelle. Nevertheless, the genericity in SSL is extended to the architecture description leading to the definition of generic links in addition to the behaviour genericity. Through SSL, we have presented the relevant features that must be supported by a language so as to represent concisely the symmetrical protocols: total genericity, object approach, high-level communication mechanisms.

The semantics of SSL is the one of Well-formed Nets. Hence, we have shown the different verification techniques which exploit the genericity: flows computation, net reductions and the symbolic reachability graph. Finally, we gave a SSL specification of the Ricart and Agrawala's algorithm which enhances the language concision.

REFERENCES

1. CCITT, "Specification and Description Language SDL. Recommendation Z.100, CCITT Blue Book, 1988.
2. ISO-IS 9074; "Estelle: a formal description technique based an extended state transition model"; 1989 (E).

3. ISO-IS 8807, "Lotos, a formal description technique for the temporal Ordering of observational Behavior"; 1989.
4. G. v Bochmann; "Usage of Protocol Development tools: the result of a survey"; Protocol Specification, Testing and Verification VII, Zurich; CH. H. Rudin (red.), North-Holland, 1987.
5. A.A.F. Loureiro, S.T. Chanson, S.T. Vuong, "FDT Tools For Protocol Development", 5th International Conference on Formal Description Techniques, Lannion, France, October 1992, Tutorials 38-78 .
6. R.E. Miller; "Protocol verification: the first ten years, the next ten years"; Protocol Specification, Testing and Verification X -IFIP, Ontario June 90.
7. S. Budkowski, P.Dembinski; "An introduction to ESTELLE: A specification language for distributed systems"; Computer network and ISDN Systems journal, vol. 14-1, 1988.
8. J.L. Richier, C. Rodriguez, J. Sifakis, J. Voiron; "Verification in XESAR of the sliding window protocol"; Protocol Specification, Testing and verification, VII ; IFIP 1987.
9. C.Y. Wang and K.S. Trivedi, "Integration of Specification for Modeling and Specification for System Design", 14th International Conference on Application and Theory of Petri Nets, Chicago, 1993.
10. G. Chiola, C. Dutheillet, G. Franceschinis, S. Haddad, "On Well-formed coloured Nets and their Symbolic Reachability Graph", in High-level Petri Nets. Theory and Application, 373-396. Springer-Verlag 1991.
11. J.M. Couvreur, "The General Computation of Flows for Coloured Nets". in Proc. 11th International Conference on Application and Theory of Petri Nets, Bonn, Germany, June 1989.
12. P. Huber, A.M. Jensen, L.O. Jepsen, K. Jensen, " Towards reachability trees for High-Level Petri Nets", Advances in Petri Nets'84, pp 215-233, 1984.
13. B. Zouari ; "Specification and Verification Methods for Communication Protocols"; Thesis of University Paris VI; December 1992.
14. S. Haddad, M. Taghelit, B. Zouari "Assessment of ESTELLE and EDT through real case studies" 13th International Symposium on Protocol Specification, Testing and Verification - PSTV XIII. Mai 1993.
15. S. Haddad "A Reduction Theory for Coloured Nets" in High-level Petri Nets. Theory and Application, 399-425. Springer-Verlag 1991.
16. K. Barkaoui, C. Dutheillet, S. Haddad "An efficient algorithm for finding Deadloks in Colored Petri Nets", 14th Conference on Application and Theory of Petri Nets, Clicago, Juin 1993.
17. G. Chiola, C. Dutheillet, G. Franceschinis, S. Haddad, "Stochastic Well-Formed Coloured Nets and Multiprocessor Modelling Applications" to appear in IEEE Transactions on Computers.
18. G. Ricart, A.K. Agrawala, "An Optimal Algorithm for Mutual Exclusion in Computer Networks". Communications of the ACM, 24(1), 9-17.