Verifying Estelle Specifications : Numerical Petri Nets Approach

A. Jirachiefpattana, R. Lai
Department of Computer Science
and Computer Engineering
La Trobe University
Victoria, Australia 3083

Abstract

The techniques for verifying protocols specified in Estelle have so far focused on translating the specifications into another form, such as finite state machines or Petri nets, for which tools of verification have already been implemented. All the Estelle verification tools impose some restrictions on the specifications to be verified: the techniques use a subset of Estelle or restrict the complexity of the specifications that can be verified; or the specifications need to be specified in a variant of Estelle, rather than standardised Estelle. In particular, the dynamic behaviours of Estelle are not handled. This paper describes the verification of Estelle specifications by translating them into those of Numerical Petri Nets, which can then be verified by a proven automated verification tool, PROTEAN.

1 Introduction

Verification is an important step to increase confidence in the final implementation of communication protocols. Simulation techniques, in which only some of the possible behaviours are explored, are not sufficient. Verification of Estelle specifications is still the subject of much research. Several techniques have been devised to verify Estelle specifications, however, these techniques have so far been based on the translation of Estelle specifications into another form, such as finite state machines, Petri nets or temporal logic formulas, for which tools of verification have already been implemented. All of the Estelle verification tools impose some restrictions on the specifications to be verified: they use a subset of Estelle or restrict the complexity of the specifications that can be verified; or the specifications need to be specified in a variant of Estelle, rather than standardised Estelle. In particular, the dynamic behaviours of Estelle are not handled. There are verification tools based on Petri nets [3,4,9,10], which have been able to verify only a verification-oriented subset of Estelle and only a static hierarchy of modules.

This paper describes the verification of Estelle specifications by translating them into those of Numerical Petri Nets (NPNs) [14,15], which can be verified by a proven automated verification tool, PROTEAN [6]. The merits of this approach, as compared to existing methods, are that no restriction needs to be imposed on the specification, specifications are fully based on standardised Estelle, all dynamic behaviours of the protocol can be handled.

2 Estelle

Estelle [1,2,7,8] is a Formal Description Technique (FDT) developed by ISO, and based on an extended state transition model. Estelle may be viewed as a set of extensions to ISO Pascal, level 0, which models a specified system as a hierarchical structure of communicating automata which may run in parallel, and may communicate by exchanging messages and by sharing, in a restricted way, some variables. With Estelle, a protocol system is specified as a hierarchically structured system of nondeterministic sequential components (instances of modules) which interchange messages (called interactions) through bidirectional links between their ports (called interaction points). Both the hierarchy of modules and the structure of links may change over time, thereby making the system a dynamic one.

3 Numerical Petri Nets

Numerical Petri Nets (NPNs), developed originally by Symons [13], are one of the many extensions of Petri nets. It has been found possible with NPNs to retain the basic principles, symbols and modes of

0-8186-3670-X/93 $03.00 © 1993 IEEE

operation of Petri nets, while adding a considerable amount of modelling convenience. NPNs, which are claimed to be a generalisation of Petri nets, overcome various limitations of Petri nets in modelling many practical systems, and also provide a more compact form of graphical representation.

NPNs consist of the following extensions :

- Tokens can have any finite number of attributes, each of which may be one of several different types;
- Global variables can be associated with the net;
- The transition enabling conditions refer not only to the tokens in the input places but also to the global variables;
- The firing of a transition not only removes tokens from the input places and inserts new tokens into the output places, but may change the values of the global variables.

NPNs have been successfully used in modelling some ISO protocols [5,12].

4 NPNs Modelling Estelle

We distinguish here the components of Estelle which can be modelled explicitly by NPNs. All module bodies and all the interaction points in the module header of any corresponding module body become places in an NPN, and transitions in the module body become transitions in an NPN. Tokens in a place denote the fact that the message and state belonging to the module instance will be processed by one of the enabling transitions in the module body. Transitions are fired to represent transitions being executed in the module body. The net result is the exchange of tokens or interactions from place to place, and a new marking is obtained. The mapping of Estelle components into NPNs is shown in the table below.

Estelle	NPN
a module body	a place
an external interaction point	a place
an internal interaction point	a place
a transition	a transition
a from-clause	an enabling condition
a provided-clause	an enabling condition
a transition block	transition operations
a when-clause	an input token
an output-statement	an output token

4.1 Tokens

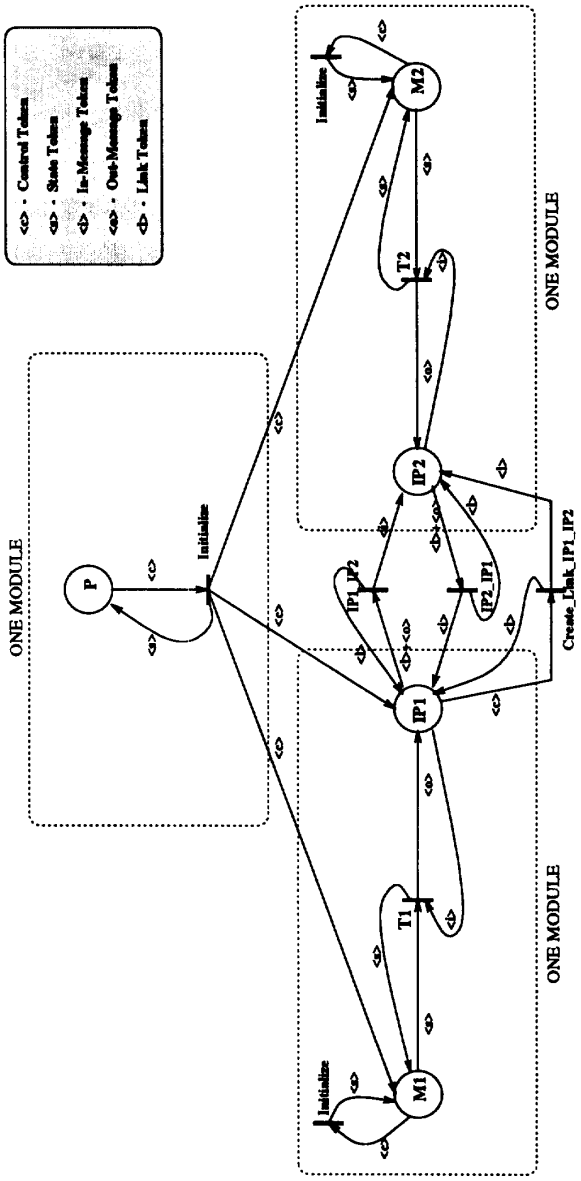


Figure 1: An example of five token types on NPN graph

In this NPN model, tokens are used to model the *dynamic behaviour* of Estelle and *interactions* exchanged between the module and its interaction point. Each token is a collection of fields or attributes. As such, the first field is used to identify what type the token is. There are five types of tokens: *control tokens*, *state tokens*, *in-message tokens*, *out-message tokens* and *link tokens*. The details of each token type are described below. Figure 1 gives an example of where each token type will appear on the NPN graph.

4.1.1 Control Tokens

This type of token is used to control the dynamic behaviour of module instances. It is a variable-length token in which the number of fields or attributes depends on the dynamic operation specified in the second field of the token. There are seven dynamic operations: *init*, *connect*, *disconnect*, *attach*, *detach*, *release* and *terminate*. The first four fields of this token are compulsory. The first field contains the string-of-character **'control'**; the second contains a dynamic operation; the third contains a module instance name which creates this token; and the fourth contains a new module instance name which is a child of the module instance in the third field. An example of this type of token is

<'control','init','PARENT','UA'>.

4.1.2 State Tokens

This type of token is used to hold the current state of the module instance and to identify which in-message token belongs to which module instance. Thus, the number of state tokens is equal to the number of module instances. A state token is a variable-length token, that is, the number of fields is not fixed, but depends on the number of variables in the initialization part of the module body definition as well as those used in the **"provided"** clause and those in the transition block. The first four fields are compulsory. The first field contains the string-of-character **'state'** used to identify the type of token; the second contains a parent module instance name; the third contains a module instance name which is a child of the module instance in the second field and possesses this state token; and the fourth contains the current state of the module instance. An example of this token is

<'state','PARENT','UA','ESTAB', ... >.

This token is created by the **initialize** transition after receiving a control token with the dynamic operation **"init"** from the parent module. This token will be preserved in the module by the output arc from the transition back to the module corresponding to the input place, as shown in figures 1 and 2.

4.1.3 In-Message Tokens

This type of token is used to carry an interaction or message from another module instance. This corresponds with an interaction on the **"when"** clause. It is a variable-length token depending on the PDU (Protocol Data Unit) of the interaction. The first four fields are compulsory. The first field contains the string-of-character **'in'**; the second contains a parent module instance name; the third contains a module instance name which is a child of the module instance in the second field; and the fourth contains the interaction followed by its PDU. An example of this token is

<'in','PARENT','UA','A_ABIN', ... >.

4.1.4 Out-Message Tokens

This type of token is used to carry an interaction or message emitted by the **"output"** statement. The format of this token is the same as for the in-message token except that the first field contains the string-of-character **'out'**. An example of this token is

<'out','PARENT','UA','A_ABRQ', ... >.

4.1.5 Link Tokens

This type of token is used to indicate that there is a link (created either by the *connect* statement or by the *attach* statement) from an interaction point of one module instance to an interaction point of another module instance. Thus, the number of link tokens is equal to the number of links between these two interaction points. This token consists of seven fields (attributes). The first field contains the string-of-character **'link'**; the second contains a parent module instance name; the third contains a module instance name which is a child of the module instance in the second field, and possesses an interaction point which is in the fourth field; the fifth contains a parent module instance name on the other side of the link; the sixth contains a module instance name which is a child of the module instance in the fifth field; and the seventh contains the interaction point belonging to the module instance in the sixth field. An example of this token is

<'link','PARENT','UA','USIP','PARENT','ACPMA','HIGH'>.

This token is created after the interaction point corresponding to the output place receives a control token

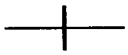
with the dynamic operation "**connect**" or "**attach**". This token will be preserved in the same way as the state token, as shown in figures 1 and 2.

4.2 NPN notation

A list of symbols and their proposed interpretation in NPNs is given below:

S_i : Place, which denotes a module or an interaction point M_i and is represented by a labelled circle

t_i : Transition, which denotes the event of transition execution and is represented by a labelled bar



<c> : Control token, which controls the dynamic behaviour of module instances;

<s> : State token, whose presence at S_i, indicates the current state and the current value of variables of a module instance of the M_i module type;

<i> : In-message token, which carries an interaction or message from another module instance;

<o> : Out-message token, which carries an interaction or message emitted by the "**output**" statement;

<l> : Link token, which indicates that two module instances and their interaction points are linked to each other by the **connect** statement or the **attach** statement;

IC : Input condition, which may be satisfied by the presence of the tokens in the associated input place;

DT : Destroyed tokens, which indicate that all tokens are removed from the associated input place when the transition fires;

CT : Created tokens, which are deposited in the associated output place when the transition fires;

|TC| : Optional transition condition, which may be satisfied by the net data variables or by values associated with the token residing in the transition's input place;

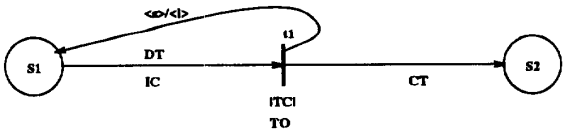


Figure 2: A simple NPN graph of an Estelle transition

TO : Optional transition operation, which may be performed when the transition fires.

A simple NPN graph of an Estelle transition is given in figure 2. The transition is enabled when both its Input Condition (IC) is satisfied (i.e. there are tokens in the place S1) and its Transition Condition (TC) is met. When the transition **t1** fires, the following events occur indivisibly and concurrently: Destroyed Tokens (DT) are removed from the input place; Created Tokens (CT) are added to the output place; the state variable in the state token is updated; and the Transition Operation (TO) on net data as a result of the execution of the Estelle transition block is performed. Note that the output arc (from the transition back to S1) creates a self-loop which is used to preserve the state token <s> (if S1 represents a module body) or the link token <l> (if S1 represents an interaction point). In addition, both the state token and the link token are used not only for keeping the state and the link of the module instance but also for identifying which message belongs to which module instance.

5 Translating Estelle Specifications into NPNs

This section describes how Estelle specifications can be translated into NPN specifications.

5.1 Modules

In Estelle, a module is defined by *a module-header definition* and *a module-body definition* associated with this header. There may be more than one *module-body definition* referring to the same *module-header definition*. Thus each pair, a *module-header definition* and an associated *module-body definition*, constitutes one Estelle module definition, or more simply, one module. Several instances of a module may be created and be simultaneously present during execution of an Estelle specification. Each one of them has the same external visibility characterised by the *module-header definition* and the same internal behaviour defined by the *module-body definition* [1].

In NPN, there will be a place for:

- each module-body identifier;
- each external interaction point identifier specified in the module-header definition; and
- each internal interaction point identifier specified in the declaration part of the module-body definition.

The parameters listed in the module-header identifier become fields in a control token having the dynamic operation "**init**". This model, however, does not include *exported variables*.

The module-initialization part of a module-body definition describes an initial control state and the way variables are initialized, and also defines an initial hierarchy and interconnection structure of descendant module instances, if any exist. This initialization part is modelled as the **initialize** transition in NPN which is fired to:

- create a state token having an initial control state and initial value of variables;
- create new module instances by sending a control token with the dynamic operation "**init**"; and
- bind interaction points by sending a control token with the dynamic operation "**connect**" or "**attach**".

The internal behaviour of a module is defined by *transitions* within the transition part of a module-body definition. Each transition is composed syntactically of two parts: *clauses* and *a transition-block*. A transition in Estelle straightforwardly becomes a transition in NPN as follows:

- A transition-block becomes a transition operation (TO) in NPN, except for an output-statement.
- The interaction point in an output-statement becomes an output place, and its associated interaction becomes an out-message token.
- The control state or stateset in the from-clause becomes a transition condition in NPN such that the state in the state token must equal the state or one of the states in the stateset in the from-clause.
- The interaction point in a when-clause becomes an input place, and its associated interaction becomes an in-message token.

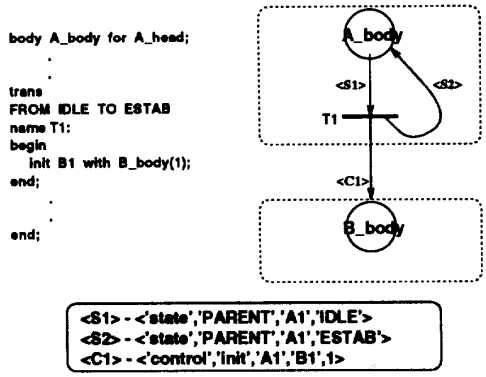


Figure 3: An output arc integrating two nets

- Boolean expressions in the provided-clause become transition conditions in NPN.
- The priority-clause and the delay-clause are not included in this model.

Note that in NPN there will be at least one more transition of each module body than in Estelle specification. This is the **initialize** transition mentioned above.

5.2 Module Composition

We can consider each Estelle module as an indivisible NPN net which consists of a place representing a module body, places representing interaction points associated with the module body, and transitions for exchanging tokens between the module body and interaction points. All of these indivisible NPN nets representing Estelle modules, however, can be integrated into one global NPN net by *an output arc* and by *three extra transitions*.

An output arc, which is used to integrate two nets, is one whose associated transition creates *a control token* as an output token and sends such a token to an output place representing a module body or an interaction point. In figure 3, there is an output arc which carries a control token having the dynamic operation "init" from the transition **T1** to the output place representing the module body **B_body**. This output arc, thus, seems to integrate two nets.

Two module instances can communicate by exchanging messages (in both directions) through a previously established communication link between their two interaction points. The link between two interaction points can be created by a *connect* or *attach* statement depending on the type of interaction point. Thus, two NPN nets representing two modules can be

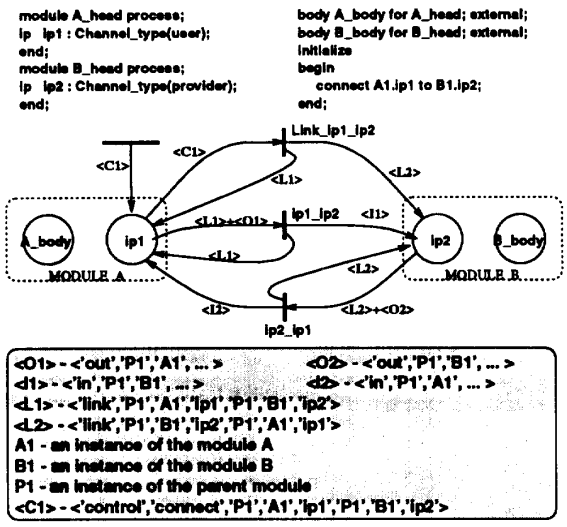


Figure 4: Three transitions integrating two nets

integrated by using *three more transitions* than those specified in the Estelle specification. The first transition is used to create *two different link tokens*: the first token is for one NPN place representing one interaction point and the second for another NPN place representing the other interaction point. The second and third transitions are used to send *message tokens* from one NPN place representing one interaction point to another NPN place representing the other interaction point. Figure 4 shows how to integrate two NPN nets using these three transitions. In figure 4, the transition **Link_ip1_ip2** is used to create two link tokens: one for **ip1** and the other for **ip2**; the transition **ip1_ip2** is used to forward message tokens from **ip1** to **ip2**; and the transition **ip2_ip1** is used to forward message tokens from **ip2** to **ip1**.

5.3 Module Instance Creation

The creation of and initialisation of a module instance in Estelle is performed with an **init** statement. Thus, an initialisation part or a transition block in Estelle which has an **init** statement will become an NPN transition, one of whose output arcs will carry a control token with the dynamic operation "init" and an actual module parameter list. The output place of this output arc will be that representing the module body specified in the **init** statement. In addition, there must be at least one more transition in the net which has a place representing such a module body. This transition will create and initialise a *state token* for a new module instance after receiving the control

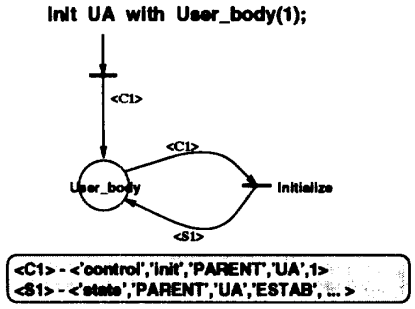


Figure 5: An NPN model of module instance creation

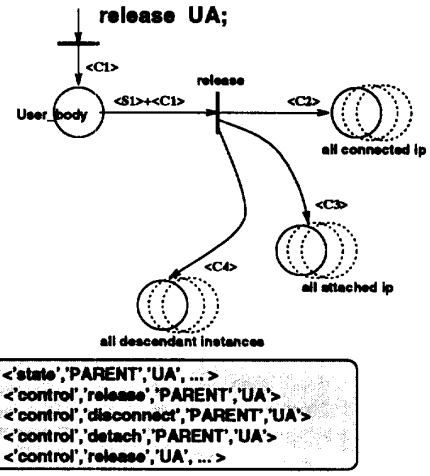


Figure 6: An NPN model of module instance release

token. It must be emphasized that the existence of a state token means the existence of a corresponding module instance. Figure 5 gives an example of modelling this.

5.4 Module Instance Release

A **release** statement is used to release module instances located at a lower hierarchy level. The result of executing a **release** statement is equivalent to the following actions: the first is to disconnect and detach all links of the module instance identified by the release statement; and the second is to destroy the module instance as well as all its descendant instances.

On using NPN, a transition block having a **release** statement will become an NPN transition, one of whose output arcs will carry a control token with the dynamic operation "release". The output place of this output arc will be that representing the mod-

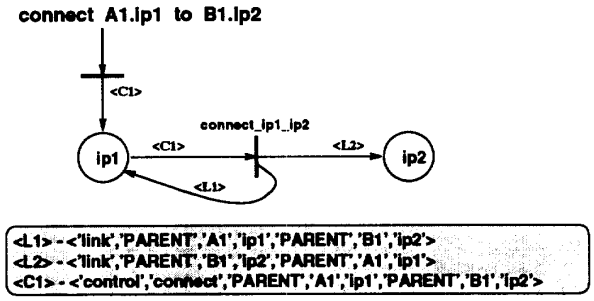


Figure 7: An NPN model of connect operation

ule body of the module instance specified in the **release** statement. In order to model the actions as the result of executing a **release** statement, there must be one more transition in the net which has a place representing the module body of the module instance being released. This transition will create control tokens having the dynamic operation "disconnect" or "detach" and send them to all NPN places representing interaction points of the module instance, and will create control tokens having the dynamic operation "release" and send them to all descendant instances. Figure 6 gives an example of modelling this. Note that the number of places representing the interaction points and the number of places representing module bodies of all descendant instances can be known from the Estelle specification.

5.5 Connect and Attach Operations

Both a **connect** statement and an **attach** statement are used to create a new link between two interaction points. Thus, an initialization part or a transition block having a **connect** or **attach** statement will become an NPN transition, one of whose output arcs will carry a control token with the dynamic operation "connect" or "attach". The output place of this output arc will be that representing the first interaction point of the first module instance specified in the **connect** or **attach** statement.

In order to create a link token for each of two NPN places representing two interaction points, there will be one more transition, whose input token will be the control token having either the dynamic operation "connect" or "attach" (depending on the type of link between these two interaction points), and whose output arc to each of two interaction points will carry a link token. Figure 7 gives an example of modelling the **connect** operation, which is the same as the **attach** operation.

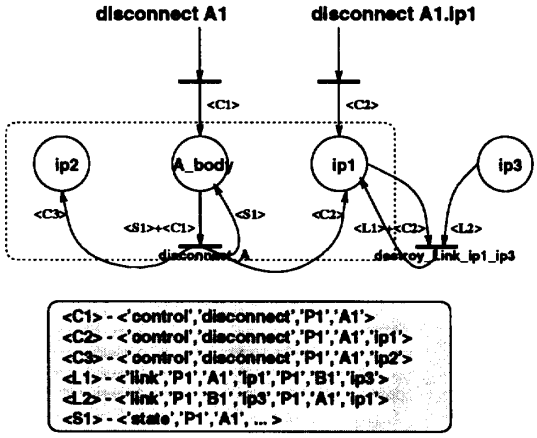


Figure 8: An NPN model of disconnect operation

5.6 Disconnect and Detach Operations

Both of these operations are used to destroy an existing link between two interaction points. Unlike the **connect** and **attach** statements, the syntax of the **disconnect** and **detach** statement has only one parameter which can be *an internal interaction point* or *an external interaction point* or *a child-external interaction point* or *a module instance*. That is, the output place of the transition creating a control token with the dynamic operation "disconnect" or "detach" can be either a place representing an external or internal or child-external interaction point, or a place representing the module body of the module instance.

In order to destroy link tokens in places representing interaction points, there will be one more transition between two places having a link. Consequently, the link tokens in these two places, which will be the input tokens of this transition, will be destroyed. In addition, in the case that the control token is sent to a place representing the module body of a module instance all of whose interaction points are required to detach or disconnect, there will be one more transition creating control tokens with the dynamic operation "disconnect" or "detach" for all places representing interaction points of the module instance. Figure 8 gives an example of modelling the **disconnect** operation, which is the same as the **detach** operation.

6 Conclusions

In this paper, we have described an approach to verifying Estelle specifications by translating them into Numerical Petri Net specifications which can then be

verified by using an existing proven automated tool, PROTEAN. There are a number of merits of this approach over existing methods. The verification is based on standardised Estelle, rather than a variant or a sub-set of it. There is no restriction on the specification styles. All dynamic behaviours of Estelle can be handled. An indivisible NPN net representing an Estelle module is very compact. Each indivisible NPN net representing an Estelle module is able to be verified individually by setting appropriate initial tokens in the place representing the module body identifier and the places representing the interaction points. This will be very useful if the size of the global net is very large.

To demonstrate the viability of our approach, we have used the ISO ACSE protocol as a case study [11], rather than the alternating bit or abracadabra protocol, as usually found in the literature. It was found to be successful. It is thus concluded that the proposed method is feasible for verifying real-life protocols specified in Estelle.

References

[1] ISO9074, Information Processing Systems - Open Systems Interconnection - ESTELLE (Formal Description Technique based on an Extended State Transition Model), 1989.

[2] ISO/IEC JTC 1/SC 21, Information Processing Systems - Open Systems Interconnection - ESTELLE - A FDT Based on an Extended State Transition Model - Proposed Draft Amendment 1: Estelle Tutorial, 1991.

[3] Azema, P., Lloret, J.C., Papapanagiotakis, G., Vernadat, F., ESTELLE Validation and PROLOG Interpreted Petri Nets, M. Diaz, J.-P. Ansart, J.-P. Courtiat, P. Azema, and V. Chari (eds.), *The Formal Description Technique Estelle*, pages 273–302, North-Holland, 1989.

[4] Azema, P., Vernadat, F., Lloret, J.C., Requirement Analysis for Communication Protocols, J. Sifakis (ed.), *in Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems, Grenoble, France*, pages 286–293, Springer-Verlag, June 1989.

[5] Billington, J., Specification of the Transport Service using Numerical Petri Nets, C. Sunshine (ed.), *Second International Workshop on Protocol Specification, Testing and Verification*, IFIP, Idyllwild, California, May, 1982.

[6] Billington, J., Wheeler, G.R., Wilbur-ham, M.C., PROTEAN: A High-level Petri Net Tool for the Specification and Verification of Communication Protocols, *IEEE Transactions on Software Engineering*, 14(3):301–316, March 1988.

[7] Budkowski, S., Dembinski, P., An Introduction to Estelle: A Specification Language for Distributed Systems, *Computer Networks and ISDN Systems*, 24:3–23, 1987.

[8] Dembinski, P., Budkowski, S., SPECIFICATION LANGUAGE ESTELLE, M. Diaz, J.-P. Ansart, J.-P. Courtiat, P. Azema, and V. Chari (eds.), *The Formal Description Technique Estelle*, pages 35–75, North-Holland, 1989.

[9] Dimitrov, V., Petkov, A., Verification Oriented Estelle Specification of Communication Protocols, R. Speth (ed.), *Research into Networks and Distributed Applications*, p.953–960, North-Holland, 1988.

[10] Dimitrov, V., Petkov, A., Verification of Specifications Written in Estelle, Using Petri Nets, *Automatic Control & Computer Science*, 23(5):20–24, 1989.

[11] Lai, R., Jirachiefpattana, A., Verification of ISO ACSE Estelle Specifications, Technical Report, Department of Computer Science and Computer Engineering, La Trobe University, 1993.

[12] Lai, R., Dillon, T.S., Parker, K.R., Application of Numerical Petri Nets to Specify ISO FTAM Protocol, In *Proceedings of the 1989 Singapore International Conference on Networks*, July 19-20, 1989, Singapore.

[13] Symons, F.J.W., *Modelling and analysis of communication protocols using Numerical Petri Nets*, PhD thesis, Dep. Elec. Eng. Sci., Univ. Essex, Telecommun. Syst. Group Rep. 152, May 1978.

[14] Wheeler, G.R., *Numerical Petri Nets - A definition*, Technical Report **REPORT 7780**, Telecom Australia, Research Laboratories 770 Blackburn Road, Clayton, Victoria, Australia 3168, 1985.

[15] Wheeler, G.R., Wilbur-Ham, M.C., Billington, J., Gilmour, J.A., Protocol analysis using Numerical Petri Nets, *In Lecture Notes in Computer Science*, 188:435–452, 1986.