Joint NCC & IIS Bull., Comp. Science, 8 (1998), 19–38

Modeling Estelle specifications using colored Petri nets*

T. G. Churina, E. V. Okunishnikova

Our aim is to validate distributed systems which are described by means of Estelle. This paper describes a procedure of translating the Estelle specifications into colored Petri nets extended by the priorities and Merlin's time concepts. The considered specifications are based on standard Estelle which developed by ISO. The hierarchy specifications with time and priorities are considered. The dynamic behaviors of the Estelle specifications cannot be handled. Thus the considered subset of Estelle provides expressive power sufficient for modeling a lot of the protocols. The nets created during the translation are semisafe nets: all places except places corresponding to arrays and interaction points can contain at most one token. The translation of all constructions is described with some informal justification.

Introduction

Two principal approaches are used for distributed system verification. The first one uses such formal description techniques (FDT) as Estelle, SDL, LOTOS, which are international standards. However, high expressive power of FDT increases difficulties of validation and verification of distributed systems. In framework of the second approach, distributed systems are modelled by finite state machines, Petri nets or their modifications, and subsequently verified. The advantage of this approach is the existence of powerful methods of analysis.

A problem of combining the advantages of both approaches is of considerable importance. Papers [8, 7] have recently examined the problem for SDL. In both the papers, new classes of Petri nets are used and methods of reachability analysis are proposed. The developed technique of the performance analysis for SDL-nets is also presented in [7]. Paper [9] describes the verification of the Estelle specifications by translating them into Numerical Petri Nets. The merit of this approach is the fact that dynamic behaviors of the Estelle specifications can be handled. The deficiencies are the fact that priority- and delay-clauses and systemprocess attribute cannot be modelled.

Our aim is to validate distributed systems which are described by means of Estelle [2, 1]. A modeling procedure has been developed for this goal. The procedure translates Estelle specification into colored Petri nets (CPN's) ex-

*Partially supported by INTAS–RFBR under Grant 95–0378.

tended by the priorities and Merlin's time concepts [10]. We choose extended CPN's to model Estelle specifications for the following reasons: a more compact graphical representation, the expressive power in concurrence and nondetermenism and, at last, the existence of automated tools for simulation and analysis. In order to retain the advantage of Petri net analysis algorithms, we consider only specifications which do not use dynamic capabilities of Estelle. A preliminary version of the translation procedure operating with the Estelle specifications which have exactly one level in the hierarchy of modules and do not use time and priorities, has been recently described in [6].

Translation of an Estelle specification to a CPN is made in several steps. At the first step, we build a CPN which represents the general structure of the Estelle specification and contains one transition for each module instance. At the second step, the body of each module is translated. At the next steps, blocks of transitions of the Estelle specification (E-transitions) are translated into CPN transitions (N-transitions).

This paper presents the extended procedure which allows to be considered the specification with time, priorities, and arbitrary hierarchy. Therefore the considered subset of Estelle provides the expressive power sufficient for modeling a lot of the protocols.

1. Basic concepts

1.1. Estelle

Estelle [2, 1] is a Formal Description Technique 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 the specified systems 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.

An Estelle specification describes a hierarchically structured system of nondeterministic sequential components which interchang messages through bidirectional links between their ports. Each component is an instance of a module defined within the Estelle specification by a module definition.

The behavior of a module and its internal structure are specified respectively by the set of transitions which the module may execute and by the definition (children modules) of the module together with their interconnection. Execution of a transition by a module is an atomic operation. Transitions of a parent module have the priority over its children's transitions.

From an external viewpoint, a module instance is a "black box". Ac-

cess in and out of that box is made with finite sets of interaction points and exported variables. Each interaction point of a module has an associated FIFO queue which receives and stores interactions received by the module through this interaction point. More than one interaction point of a module may share the same queue. A channel is associated with each interaction points which defines two sets of interactions. These sets consist of interactions which can be transmitted and received, respectively, through the interaction points. External access of exported variables is restricted exclusively to the parent module instance.

A module definition in Estelle may textually include definitions of other modules. This leads to a hierarchical tree structure of module definitions. The hierarchical tree structure of modules constitutes a pattern for any hierarchy of module instances. The hierarchical position of a module instance corresponds to the position of the module definition in this pattern.

A module may have one of the following class attributes: systemprocess, systemactivity, process, activity or may be not attributed at all. The modules attributed with systemprocess or systemactivity are called system modules. Attributes define two possible forms of execution. The *process* and *activity* attributes represent synchronous parallel execution and nondeterministic sequential execution respectively.

1.2. Colored Petri nets

A colored Petri net model is an extension of the basic Petri net model. It consists of three different parts: *the net structure*, *the declarations* and *the net inscriptions.*

The net structure is a directed graph with two kinds of nodes, *places* and *transitions*, interconnected by *arcs* in such a way that each arc connects two different kinds of nodes. Places and their *tokens* represent states, while transitions represent state changes. An arc represents an input or output relationship between a place and a transition.

The declarations contain definitions of *color sets* and declarations of variables which can be bound by colors. The declarations can also contain definitions of new operations and functions which can be applied to the colors. Moreover, a color set definition often implicitly introduces operations and functions which can be applied to its members.

Each *net inscription* is attached to a place, a transition or an arc. In the CPN each place has three different kinds of inscriptions: *name*, *color set* and *initialization expression. The color set* determines the type of tokens which may reside on the place. *The initialization expression* determines *the initial marking* of the place. Transitions have two kinds of inscriptions: *names* and *guards. The guard* of a transition is a boolean expression which must be fulfilled before a transition can occur. The arcs have one kind

of inscriptions: *arc expressions*. They may contain variables, constants, functions and operations. When the variables of an arc expression are bound (i.e., replaced by colors), the arc expression must evaluate to a color which belongs to the color set attached to the place of the arc.

Transitions are active components of the CPN's. If transition is *enabled*, it may occur. Before a transition may occur, all the variables of the guard of the transition and the arc expressions on its surrounding arcs have to be bound to colors of the corresponding types. The transition is *enabled* for this binding if the guard evaluates to true and each of the input places contains at least the tokens to which the corresponding arc expression evaluates. *The occurrence* of the transition removes tokens from their input places and adds tokens to their output places. The number of removed/added tokens and the colors of these tokens are determined by the value of the arc expressions evaluated with respect to the binding.

1.3. Extension of CPN's with time and priorities

The computational model for Estelle is formulated in time-independent terms: one of the principal assumptions of this model is that nothing is known about the execution time of transitions. However, some Estelle transitions may contain a delay-clause. The intention of the delay-clause is to indicate that a transition's execution should be delayed. So, time extension of CPN's is necessary to model specifications whose behavior depends on explicit time parameters. In [4], the CPN model is extended with a time concept represented by a global clock and time stamps of tokens. However, the proposed time concept is not convenient for modeling delay-clauses of E-transitions. We define own time extension of CPN's using the model proposed by Merlin [10]. Our choice is conditioned by a similarity of time concepts.

Merlin's time Petri nets use delays specified by an interval. In [10] each transition is associated with a pair of nonnegative numbers $d_{\min}$ and $d_{\max}$ representing the minimum and the maximum delay time. The enablement of the transition means that it is enabled in the usual Petri net sense. Some transitions may be enabled by a marking, but not all of them may be allowed to fire due to the firing constraints of transitions.

The value $d_{\min}$ gives the minimal time that the transition must be enabled before its firing. The value $d_{\max}$ denotes the maximum time during which the transition can be enabled without being fired. Values $d_{\min}$ and $d_{\max}$ are relative to the moment at which the transition becomes enabled. The origin of time scale is associated with the initial marking. If the transition becomes enabled at time τ, then the transition can fire in the interval $[\tau + d_{\min}, \tau + d_{\max}]$ unless it is disabled before $\tau + d_{\max}$ by the firing of another transition. The interval $[d_{\min}, d_{\max}]$ is called *a firing interval* of the

transition. In the model there is a nondeterminism about the choice of the firing time within the firing interval. The transition may fire at any time in its firing interval. The firing of the transition is an instantaneous event and "takes no time".

We integrate net models straightforward adding Merlin's time model to CPN's. Transitions become the new kind of inscriptions — the firing interval. A transition is enabled in a marking in the integrated net if and only if it is enabled in the underlying color net. An enabled transition may fire if it is allowed to fire due to the firing constraints. Such transition is called the *firable* transition.

Let us consider the colored net extended with time which is given in Figure 1. The transitions $t1$ and $t2$ do not have guards, their firing depends on time constraints only. The transition $t3$ is enabled if the token at the place *count* is not larger than 3. Thus, all transitions are enabled by the initial marking. However, the transitions $t1$ and $t3$ are only firable from initial state. The transition $t2$ is not firable since $t1$ fire at the latest at time 1. Occurrence of the firing sequence $t1$, $t1$, $t1$, $t1$ leads to the state at which both the transitions $t1$ and $t2$ are enabled and firable. The guard of the transition $t3$ does not fulfilled, so $t3$ is not enabled.

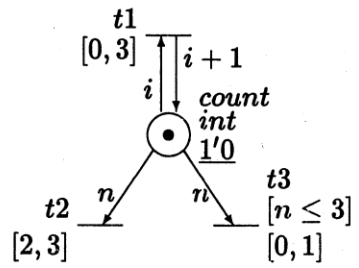

Figure 1. Example of time colored net

The clause-groups of some Estelle transitions may contain also a priority-clause. Modeling such Estelle transitions requires priorities to be added to the CPN model. Net transitions obtain a new kind of the inscriptions: *priority*. The priority is a nonnegative integer and taken into account while selecting firable transitions from the set of enabled transitions. The lowest nonnegative integer is the highest priority. If the priority is omitted, the lowest priority is assumed. The enabled transition may fire if the priority of any other enabled transition is not less than the priority of the selected transition.

If we consider both extensions, the firable transitions in the net are defined in the following way. The transition is firable if it is allowed to fire due to the time constraints and it has the priority not larger than the

priority of any other enabled transition. Such definition of firable transition coincides with the firing rule in Estelle.

2. Structure of communication links

Let us consider the first step of the net construction. The translation principle at the first step is the same as for the specifications with exactly one level in the hierarchy of modules. The declaration and initialization parts, module headers and a part of information from module bodies are used.

There is one N-transition corresponding to every module variable independently of module instance position in the hierarchy of the Estelle specification. At this step, classes of module instances are unessential. The module classes affect a computation steps organization which is considered below. The name of the module instance is used in our paper as the N-transition name. N-transition guards are not defined because the N-transitions corresponding to the module instances will be replaced by subnets during the further construction.

The net declarations should contain color sets which represent Pascal data types. Declarations of user data types are also translated into color sets. In the further net construction, the net declarations are supplemented with variables included in arc expressions and guards of the N-transitions and, sometimes, with new color sets.

Interaction points of module instances are represented by places which are connected with the N-transitions corresponding to the module instance. The mapping of interaction points is made in the following way:

- if the module instance only receives messages through an interaction point, then the interaction point becomes the one place. This place is an input place for the N-transition corresponding to the module instance;
- if the module instance only sends messages through an interaction point, then the interaction point becomes one output place of the corresponding N-transition;
- if the module instance both sends and receives messages through an interaction point, then two places correspond to this point. One is an input place and another is an output place of the corresponding N-transition;
- all input places which correspond to interaction points with the common queue are combined (united into one place).

The places which correspond to connected interaction points are combined in such a way that the input place of one N-transition is combined with the output place of another N-transition.

Translation of interaction points included in a link is different from translation of the connected interaction points. The places which correspond to the connection endpoints are combined in the net. The input places corresponding to the interaction points which are not connection endpoints should be deleted. The output places which correspond to the intermediate interaction points are kept in the net, but they are not combined with any other place.

Declarations of interactions are transformed into color sets in net declarations. The created color sets are used by modeling queues of messages. We use the enumeration of tokens for representing of FIFO queue (Figure 2), because all tokens in a place have equal status. The token value at the place "Turns" belongs to the set of color defined as follows:

$$color \quad FIFO\text{-}counter = product\ Int * Int.$$

At the initial marking the place *Turns* contains the token $\langle 0, 0 \rangle$. The first field of the token at the place *Turns* is the number of token at the place *Queue*. This number corresponds to message at the queue head. The second field is the number of the next token to be placed at *Queue*. This token corresponds to message from the queue tail.

The place *Turns* is the input and output place for any net transition removing or adding the token to the queue place. At the transition *in* firing the token with the value $\langle h, t \rangle$ is removed from the place *Turns* and the token with the value $\langle h, t+1 \rangle$ is added to this place. The new token $\langle t, d \rangle$ is added to the *Queue* place and its number is maximum among all the tokens from the same place. The value of the second field token at the place *Turns* is increased by 1. This value is the number of the next token to be added to the place *Queue*.

At the transition *out* firing the token $\langle h, t \rangle$ is replaced by the token $\langle h+1, t \rangle$ at the place *Turns*, and the token $\langle h, d \rangle$ is removed from the place *Queue*. Note that the token removed from the place *Queue* is uniquely defined by the first field value of the token at the place *Turns*.

Each formal parameter of the module instance is represented by one place. This place is an input and output place of the N-transition corresponding to the module instance. The color set of the place which represents the formal parameter is a type of the parameter. The initial marking of the place is determined by the initialization part of the specification.

Export variables are translated into places similar to the formal parameters. The difference is in the following. The place corresponding to export variable is an input and output place of two N-transitions. The first one represents the module instance which contains declaration of the variable. The second N-transition represents the parent instance.

We use the following elementary example to illustrate the first step of translation. The specification contains one channel definition and two mod-

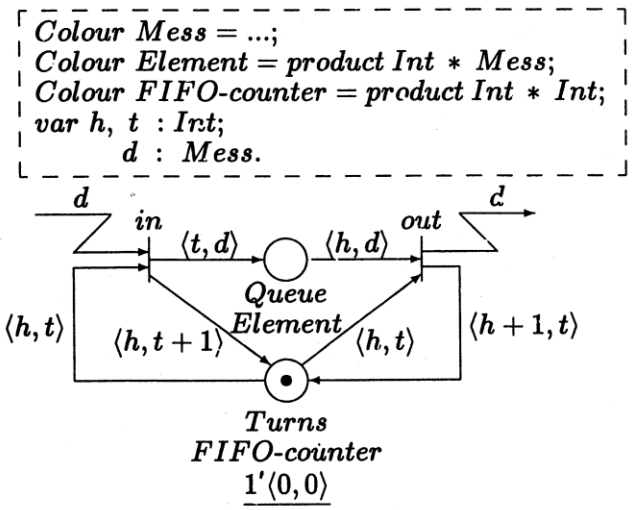

Figure 2. Modeling FIFO queue

ule definitions. Both modules *sender* and *receiver* are attributed as systemprocess. Each module has two interaction points. Individual queues are associated with interaction points. The initialization part specifies the system which consists of the instance *ms* of the module *sender* and the instance *mr* of the module *receiver*. The interaction points $s1$ and $s2$ of the module instance *ms* are connected with interaction points $r1$ and $r2$ of the module instance *mr*, respectively.

```
specification example;
channel CONN(ROLE1,ROLE2);
by ROLE1, ROLE2: mess(i:integer);

module sender systemprocess;
 ip s1:CONN(ROLE2) individual queue;
    s2:CONN(ROLE1) individual queue;
end;

body send for sender;

  state  wait, deliver;
  var j : integer;
    initialize to deliver
     begin  j:=0  end;

trans (*transmission of a new message*)
```

```
      from deliver to wait
      begin  output s2.mess(j) end;
trans (*reception of acknowledgement *)
      from wait to deliver
      when s1.mess(i) provided (i=j)
      begin j:=j+1 end;
end;

module receiver systemprocess;
 ip r1:CONN(ROLE1) individual queue;
    r2:CONN(ROLE2) individual queue;
end;

body reception for receiver;

var  j:integer;
initialize  begin j:=0 end;

trans (* reception of message and transmission *)
      (* of acknowledgement *)
     when r2.mess(i) provided (i=j)
     begin output r1.mess(j); j:=j+1 end;
end;

modvar  ms: sender;
    mr: receiver;
initialize  begin
   init  ms with send; init mr with reception;
   connect ms.s1 to mr.r1; connect ms.s2 to mr.r2;
end;
end.
```

Figure 3a demonstrates N-transitions which correspond to the modules MS and MR. One place corresponds to each interaction point. Figure 3b shows a CPN which is obtained by combination of the connected interaction points. The net declarations are omitted.

3. Body of module instance

There may be more than one definition of a module body associated with a given module header. At the first step of the translation, all N-transitions corresponding to the instances of the same module have the same external visibility characterized by places connected with these N-transitions. The

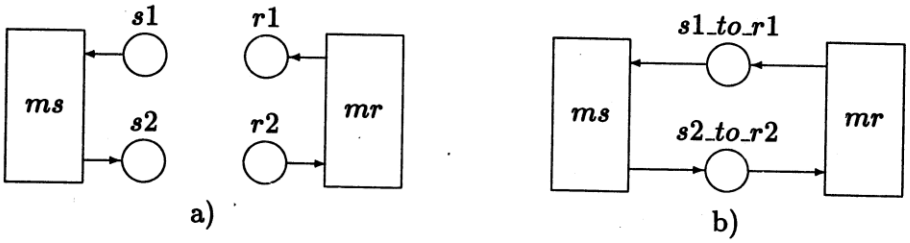

Figure 3. Net modeling specification *example* (top level)

choice of the body for the module instance is defined by the initialization part of the specification. At the second step, each N-transition corresponding to a module instance is replaced by the net which represents the body of the module. If the definition of the module body contains the keyword *external*, substitution of the subnet is not made.

The construction of the subnets for the module body does not depend on position of module instance in the hierarchy of the specification. The declaration and initialization parts of module body and part of information from E-transition declarations are used during the subnet construction.

The subnet contains one N-transition for each simple E-transition of the module instance, i.e., for each E-transition which has one transition block. Then all nested E-transition are expanded to simple E-transitions before translation.

All data types defined in the declaration part of the module body as well as a set of local states (if it exists) are transformed into color sets.

The subnet contains also one place for each variable (including arrays). The color set of the place which represents a variable is a type of the variable. The color set of the place representing an array is a set of pairs which consist of an index and a value of a corresponding array element. The initial marking of all places is determined by the initialization part of the module. If a module has a set of local states, then there is a place *State* in the net. The set of local states defines its color set. Otherwise there is a place *Manager* in the net which contains a token carrying no information ("uncolored" token).

At the second step, each place corresponding to the interaction point obtains the auxiliary place which contains a token-counter for FIFO queue (see Figure 2).

The interconnection of places and transitions is determined as follows.

- if some variable or array is referred in an E-transition, then the corresponding place is an input and output place of the N-transition associated with this E-transition;
- if some E-transition is an *input* transition (i.e. it contains *when*-clause), then there is an arc from a place which represents the in-

teraction point referred in the when-clause of the E-transition, to the corresponding N-transition; the auxiliary place is simultaneously input and output place for the N-transition;

- if the E-transition is an *output* transition (it contains the statement *output*), then there is an arc from the corresponding N-transition to a place associated with the interaction point. The auxiliary place is simultaneously an input and output place for the N-transition;
- if there exists the place *State*, then it is an input and output place for all N-transitions, otherwise the place *Manager* is the input and output place for all the N-transitions.

The N-transitions being created at this step are "black boxes" as at the first step.

We use the module *sender* of the specification *example* to illustrate the translation of the module body.

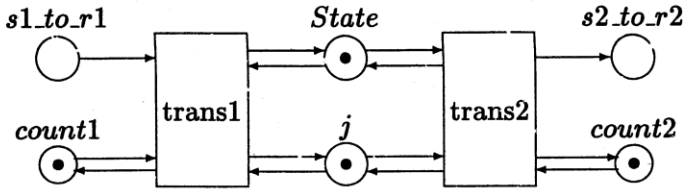

Figure 4. modeling body of module *sender*

The net (see Figure 4) contains the place *State* with the color set {*deliver*, *wait*}. The place *j* represents the variable *j*. The net has the places *s1_to_r1* and *s2_to_r2* corresponding to the connected interaction points. Moreover, there are two auxiliary places *count1* and *count2* which contain a information about the organization of the FIFO queues in the places *s1_to_r1* and *s2_to_r2* respectively.

The net has two N-transitions *trans1* and *trans2* corresponding to the E-transitions. Both places *State* and *j* are input and output places of *trans1* and *trans2*. The place *s1_to_r1* is an output place of *trans1*, because the first E-transition is output one. The place *s2_to_r2* is an input place of *trans2*, because the second E-transition is input one.

4. Mapping of Estelle transition

At the third step, the translation of the E-transitions is made. For the present, we leave out of consideration the time aspect and will return to this problem later.

4.1. Block of Estelle transition

The third step is the last step of the translation, if the E-transition satisfied the following conditions:

- the E-transition does not contain loops or calls of procedures and functions;
- there are no statements in the E-transition block which use variables changed during the execution of this E-transition;

In this case, creation of the N-transition corresponding to the E-transition is completed. The guard and the expressions on the surrounding arcs of the N-transition are defined. The guard is a combination of the from- and provided-clauses. If the E-transition is the input transition, the guard includes also the condition that the token removed from the corresponding place is the head of the FIFO queue. The to-clause and the block of the E-transition define the arc expressions.

The second transition of the module *sender* of the specification *example* is an example of the E-transition whose translation is completed at the third step. Figure 5 represents the net which models this E-transition.

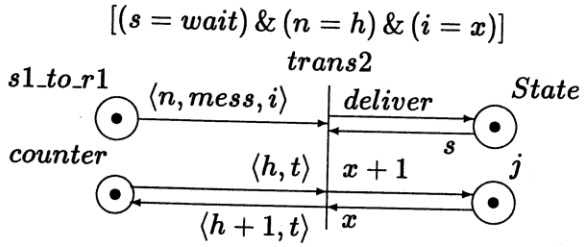

Figure 5. Net modeling the second E-transition of the module *sender*

The translation process continues if the E-transition does not satisfy the above conditions. The block of the E-transition is divided into subblocks. Each subblock is represented by the N-transition. The N-transitions are connected consequently by means of the connective places. Each subblock is translated separately. In turn, the subblock can be divided into subblocks during the translation. The block decomposition continues until each subblock can be represented by one N-transition. Then we can speak about the first and last N-transitions of the net modeling the E-transition or the subblock. The rule of the interconnection given at the second step is applied to each resulted E-transition.

The procedure/function definition is translated into the net by the same way as the blocks of the E-transitions. The resulted net is substituted for the call operators. We use the library net fragments to model the standard language statements such as *if*, *for*, *while*, *repeat*. Figure 6 gives an example of modeling the following E-transition.

```
from wait to deliver
when s1.mess(i)
provided i-j > 0
    begin for n = j to i rec[n mod k] := 1; end;
```

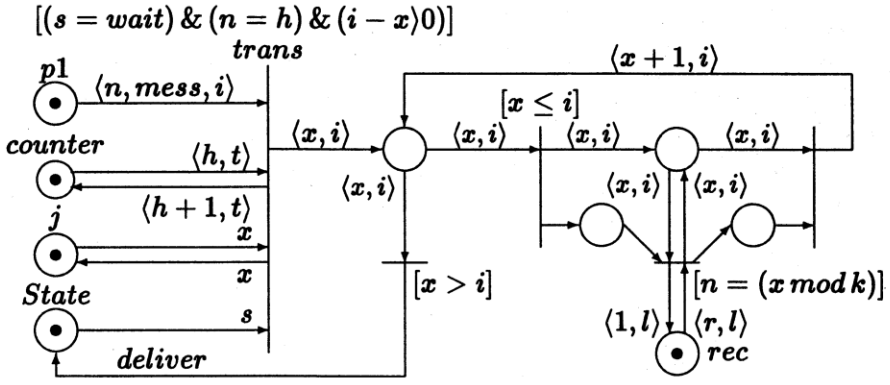

Figure 6. Net modeling *for* loop

The provided-clause including the function call is represented by the separate subblock. Otherwise, the provided-clause is the guard of the first N-transition of the net which models the E-transition.

The place *State/Manager* is the input place of the first N-transition and the output place of the last N-transition of the modelled net, if the E-transition is represented by more than one N-transition. If modeling the execution of the E-transition is started, then none N-transition which models the other E-transition, can fire. Modeling is completed by firing the last N-transition which returns the token into the place *State/Manager*. Thus, the atomicity of the E-transition execution is guaranteed.

4.2. Delay- and priority-clauses

Let us return to the time aspect. The delay-clause of the E-transition can have three different syntax forms: $delay(d1)$, $delay(d1, d2)$ and $delay(d1, *)$. The delay-clause $delay(d1)$ is semantically equivalent to $delay(d1, d1)$. The intention of the delay-clause is to indicate that the execution of the E-transition should be delayed. The value $d1$ denotes the minimum time which the E-transition must remain enabled until it may be offered for execution. The value $d2$ is the maximum time the E-transition may be delayed. The delay-clause $delay(d1, *)$ means that the delay time has no upper bound.

If the specification contains the E-transition with the delay-clause, then all E-transitions are modelled by more than one N-transition. It is possible to say that the modelled net of each E-transition consists of three parts. The

first one represents the provided-clause. The second part is the auxiliary subnet which models the delay of the E-transition. The last part represents the block of the E-transition.

Let us consider the auxiliary subnet for the delay-clause of the form $delay(d1, d2)$ presented at Figure 7. We give some explanations to this picture. The places *enbl* and *switch* have the color set $\{0, 1\}$. Both places initially contain the tokens with the value 0. The place *fir* can contain uncolored token. The delay of the E-transition is determined by the N-transition *timer*. This N-transition is connected only to the places *switch* and *fir* and has the firing interval $[d1, d2]$. All the other N-transitions of the modelled net have the firing intervals $[0, 0]$.

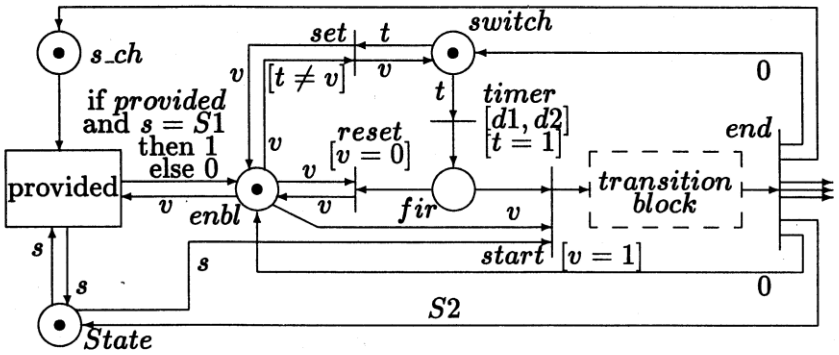

Figure 7. Net modeling the E-transition with delay-clause $delay(d1, d2)$

The box *provided* in Figure 7 denotes the subnet which models the provided-clause of the E-transition. When the E-transition is newly enabled, the token 0 at the place *enbl* is replaced by the token 1 after the execution of this subnet. After that the N-transition *set* fires. The token 1 appears at the place *switch* and the N-transition *timer* becomes enabled. Firing the N-transition *timer* is allowed between $d1$ and $d2$. It adds the token to the place *fir*. If at this moment the place *enbl* contains the token 1, the N-transition *start* can fire. Firing the N-transition *start* begins the execution of the net modeling the block of the E-transition.

Denote the considered E-transition by $E1$. If the execution of the net modeling the other E-transition (let $E2$) occurs, the re-evaluation of the provided-clause of $E1$ is made. Further "the execution of $E1$ ($E2$)" means the execution of the corresponding subnet. Let us assume that the N-transition *timer* did not fire. There are two possibilities. If the execution of $E2$ does not disable $E1$, the token at the place *enbl* keeps the value 1. The N-transition *timer* remains enabled, i.e., the timer of $E1$ remains turned on. If $E2$ disables $E1$, then the execution of the provided-clause of $E1$ replaces the token 1 at the place *enbl* by the token 0. The N-transition *set* is enabled in the resulted marking. It fires and disables the N-transition *timer*.

There are also two possibilities, if the execution of $E2$ is completed after firing the N-transition *timer*. Remember that the execution of $E1$ cannot begin before the completion of the execution of $E2$, because the place *State/Manager* is empty. The check of the provided-clause and the N-transition *start* are simultaneously enabled after the completion of $E2$. We assign the maximum priority to the first N-transition of the net modeling the provided-clause. Thus, the re-evaluation of provided-clause is always made before the execution of the E-transition can be started. The place *s_ch* guarantees that the re-evaluation of the provided-clause is made only after the execution of some E-transition. If $E1$ remains enabled after the execution of $E2$, the place *enbl* keeps the token 1. The N-transition *start* is firable. Otherwise, the token 1 at the place *enbl* is replaced by the token 0. After that the N-transition *start* is disabled, but the N-transition *reset* is firable. Its firing disables the net modeling $E1$.

The re-evaluation of the provided-clause determines the enablement of the E-transition. We assign the maximum priority to the first N-transition of the net modeling the provided-clause because the start of the check of the provided-clause and the N-transition *start* can be simultaneously enabled. Thereupon, the execution of the nets modeling only the enabled E-transitions is possible. The place *s_ch* guarantees that the re-evaluation of provided-clause is made only after the execution of some E-transition.

Observe that the delay of the E-transition is modelled by the N-transition isolated from the places which can be used by the N-transitions corresponding to the other E-transitions. Therefore, the E-transition turning on/turning off depends only on the validity of its enabling clauses.

The modelled nets of the delay-clauses of the forms $delay(d1)$ and $delay(d1, *)$ differ from the considered case by firing intervals. The N-transition *timer* has the firing interval $[d1, d1]$ in the auxiliary subnet modeling the delay-clause $delay(d1)$. In all the other respects the net is the same. The delay-clause $delay(d1, *)$ is represented by the firing interval $[d1, d1]$ of the N-transition *timer* and by the firing interval $[0, \infty]$ of the N-transition *start*.

Let us consider the translation of the E-transitions with the priority-clauses. The priority is assigned to the modelled N-transition or to the first N-transition of the modelled net (if the E-transition is modelled by more than one N-transition) in the specifications which do not contain the E-transitions with the delay-clauses. Otherwise, the priorities are assigned to the N-transitions *start* of the auxiliary subnets in such a way that the first N-transition of the net modeling the provided-clause keeps the maximum priority.

5. Organization of computation step

The behavior of hierarchical specifications depends on the classes which are assigned to module instances. Translation of such specifications requires the additional structures to be constructed. These structures model the organization of the computation step. They are building when construction of the modelled net for each module instance has been completed. The construction of the structures completes the translation of the specification. There are two different kinds of structures. They are used for the module instances attributed as *(system)activity* and as *(system)process*, respectively. The separate structure is constructed for each subsystem. It connects the parent instances with their children instances inside the subsystem.

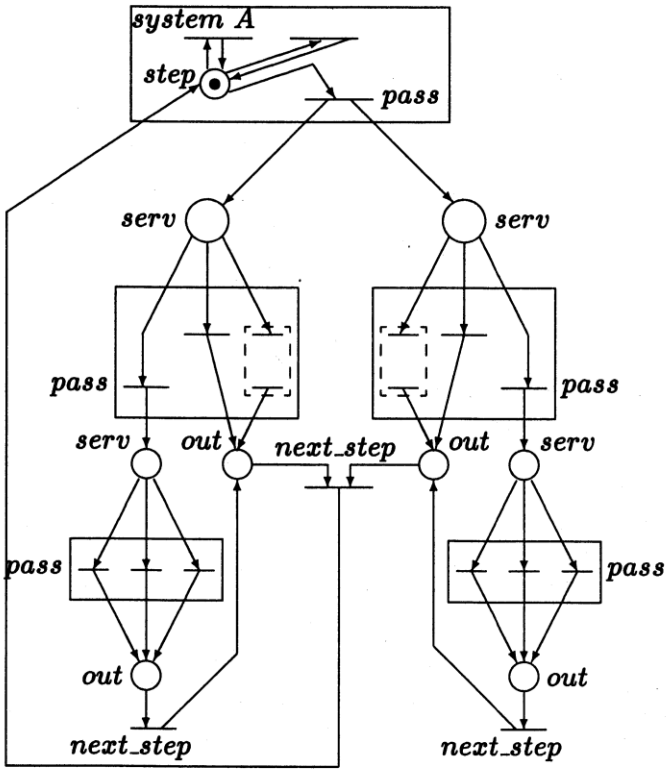

Figure 8. Modeling the computation step

Figure 8 gives the additional structure for the subsystem rooted at the instance of systemprocess. The structure models the parallel execution of all selected E-transitions. For this goal, the place *step* and the transition *pass* are created in the system module. In addition, the N-transition *next_step* is created for each module instance which has the children instances attributed

as *process*. Also, each subnet modeling the process instance contains one N-transition *pass* and two service places *serv* and *out*. If the E-transition of the process is represented by one N-transition, the places *serv* and *out* are input and output places, respectively, for this N-transition. If several N-transitions correspond to the E-transition, the place *serv* is an input place for the first N-transition and the place *out* is an output place for the last N-transition of the subnet which models this E-transition.

The input place of the N-transition *pass* is the place *serv* connected to the considered module instance. The output places of *pass* are all the places *serv* connected to the children instances of the considered process. The place *out* is the output place of the N-transition *pass*, if the module instance does not have children. The N-transitions *pass* are used to model the parent/children priority. The execution of the subnets modeling the children instances is possible after firing the N-transition *pass* corresponding to their parent.

The execution of the subnet modeling the E-transition from some module instance adds the token to the place *out* corresponding to this instance. If all the places *out* corresponding to the module instances with the same parent contain the tokens, the N-transition *next_step* is firable. If the parent is the system instance, firing *next_step* adds the token to the place *step*. Otherwise, the token is added to the place *out* of the parent instance. The next computation step is possible when the place *step* contains the token.

Let us outline the differences of the additional structure for the subsystem rooted at the instance of systemactivity. One place *serv* is connected to all the subnets modeling the module instances with the same parent. One place *out* is connected to all the subnets modeling the module instances of the same nesting level.

Observe that the similar way of the construction is used, if the process instance has the children instances attributed as *activity*. In this case the process instance plays the role of the system instance.

6. Net size estimation

Let us consider the particularities of the resulted nets. It is clear that all the places except representing arrays and FIFO queues can contain at most one token. In addition, the nets are "dense". It means that most of the places have a large number of surrounding arcs because a lot of the E-transitions use the same variables. Also, all subnets modeling the E-transition use the place *State/Manager*.

Now we give some estimate of the net size. We suppose that the module instance contains M variables including *var* local variables, *arr* arrays, *ip* interaction points, *par* parameters, N statements including k_1 for-loops, k_2

while-loops, k_3 repeat-loops, k_4 if-statements, D procedure and function definitions, C call statements of procedure and function. Let us denote

$k = k1 + k2 + k3 + k4$ and

$att = var + arr + 1 + 4 * ip + par$.

If the module instance does not contain the delayed E-transitions, then the resulted net has at most

$TN = (N + 4 * k) * (C * D + 1)$ transitions and

$PN = (att + 4 * k) * (C * D + 1) + N$ places.

At most 5 places and 4 N-transitions are added to the net per each delayed E-transition. Priorities do not increase the net size.

The net modeling specification has PS places and TS N-transitions. Let l be the number of the module instances in the specification. Then PS is the sum of the number PM for each module instance and $2 * l$ service places of the additional structure. TS is the sum of the number TM for each module instance and $2 * l$ additional N-transitions.

This is an upper bound. The resulted net has a smaller size. If the specification does not contain procedures and functions, the estimate is linear.

Conclusion

The paper represents the procedure of translation of the static Estelle specifications into extended colored Petri nets. The nets created during the translation are semisafe nets: all places except places corresponding to arrays and interaction points can contain at most one token. All constructions are described with some informal justification. It is obvious that the formal justification will be cumbersome. The paper gives an upper bound of the size of the created nets.

This work is a part of the large project aimed at validation of distributed systems, in particular, communication protocols. At present, development of the program system is completed. The system consists of two parts. The first one automatically constructs the modelled net for the Estelle specification using the described procedure. The second part is the extended NetCalc system [6] which allows us to edit and simulate the nets being created. A number of protocols were used when developing the system of the automatic net construction, for example, sliding window [11, 12], InRes [13] and ring [14] protocols.

In future, we intend to develop formal justification. The other goal is application of Design/CPN system [5] to the nets created during the translation in order to validate complex protocols.

Acknowledgements. We would like to thank our scientific supervisor Valery Nepomniaschy for his permanent attention and discussions which have allowed this paper to be improved. We would also like to thank Paul Chubarev

whose constructive comments during implementation of the presented procedure were helpful.

References

[1] S. Budkowski, P. Dembinski, *An introduction to Estelle: a specification language for distributed systems*, Computer Networks and ISDN Systems, 1988, No. 14, 3–23.

[2] *ISO 9074*, Information Processing Systems – Open Systems Interaction – ESTELLE – A Formal Description Technique Based on an Extended State Transition Model, 1989.

[3] K. Jensen, *An introduction to the theoretical aspects of colored Petri nets*, LNCS, Springer–Verlag, Berlin, 1994, **803**, 230–272.

[4] K. Jensen, *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*, Monographs in Theoretical Computer Science. An EATCS Series, Springer–Verlag, **2**, Analysis Methods, 1996.

[5] K. Jensen, S. Christensen, P. Huber, M. Holla, *Design/CPN. A reference manual*, Meta Software Corporation, 125 CambridgePark Drive, Cambridge MA 02140, USA, 1991.

[6] V.A. Nepomniaschy, G.I. Alekseev, A.V. Bystrov, T.G. Churina, S.P. Mylnikov, E.V. Okunishnikova, *Petri net modeling of Estelle specified communication protocols*, Proc. 3rd Int. Conf. Parallel Computing Technologies, LNCS, Springer–Verlag, Berlin, 1995, **964**, 94–108.

[7] F. Bause, H. Kabutz, P. Kemper, P. Kritzinger, *SDL and Petri net performance analysis of communicating systems*, Proc. IFIP fifteenth intern. Symp. on protocol spec., test. and verif., Warsaw, 1995, 259–272.

[8] J. Fischer, E. Dimitrov, *Verification of SDL'92 specifications using extended Petri nets*, Proc. IFIP fifteenth intern. Symp. on protocol spec., test. and verif., Warsaw, 1995, 455–458.

[9] R. Lai, A. Jirachiefpattana, *Verifing Estelle protocol specifications using numerical Petri nets*, Computer Systems Sci & Eng, 1996, **1**, 15–33.

[10] B. Berthomieu, M. Diaz, *Modelling and verification of time dependent systems using time Petri nets*, IEEE Trans. on Software Eng., 1991, **17**, No. 3, 259–273.

[11] N.V. Stenning, *A data transfer protocol*, Computer Networks, 1976, No. 1, **99**–110.

[12] J.L. Richier, C. Rodriguez, J. Sifakis, J. Voiron, *Verification in XESAR of the Sliding Window Protocol*, Proc. IFIP Inter. Symposium on Protocol Specification, Testing and Verification VII, North–Holland, 1987, 235–248.

[13] B. Ferenc, D. Hogrefe, A. Sarma, *SDL with applications from protocol specification*, Carl Hanser Verlag and Prentice Hall International (UK) Ltd., 1991.

[14] R. Cohen, A. Segall, *An Efficient Reliable Ring Protocol*, IEEE Transaction on Communications, 1991, **39**, No. 11, 1616–1624.