Technical Report

Specification of an Executor of Extended Simple Colored Petri Nets

Antonio Camurri and Alessandro Coglio

Dept. of Informatics, Systems, and Telecommunications (DIST)
Faculty of Engineering – University of Genoa

Viale Causa 13
16145 Genova, Italy

E-mail: {music, tokamak}@dist.unige.it

December 1997

Specification of an Executor of Extended Simple Colored Petri Nets

Antonio Camurri and Alessandro Coglio

Abstract

We present a formal specification of an executor of Extended Simple Colored Petri Nets (ESCP-nets), meant to be externally supervised. The execution of an ESCP-net proceeds through discrete steps, each performed in response to a call from an external supervisor, in charge of indicating which controlled transitions are allowed to fire and which tokens can be assigned to their controlled variables. We have realized an object-oriented C++ implementation of the executor, as part of an industrial project for Demag-Italimpianti (the largest Italian industry producing plants). After giving the specification of the executor, we also discuss some implementation issues.

1. Introduction

In this report we present a formal specification of an executor of Extended Simple Colored Petri Nets (ESCP-nets) [3, 2, 1]. We have developed an object-oriented C++ implementation of this executor, as part of an industrial project for Demag-Italimpianti (the largest Italian industry producing plants). The project consisted in the realization of software tools for the development and execution of plant simulators built according to an architecture [1] where the executor is supervised by means of declarative rules about the marking of the ESCP-net. In fact, the executor we specify here is meant to receive information (from an external supervisor) about which controlled transitions are allowed to fire and which tokens can be assigned to their controlled variables.

In Section 2 we present our specification of the executor, and in Section 3 we discuss some implementation issues. The formal definitions we give in Section 2 are based upon those in Section 2 of [3]. The mathematical concepts and notations we use are explained in the Appendix of this report, as well as in the Appendix of [3].

2. Specification

The execution of an ESCP-net by our executor proceeds through discrete steps at evenly spaced time instants of the simulated time axis (which does not necessarily coincide with the real time axis). The distance between two successive such time instants equals the unit of waiting times of tokens. The executor performs each execution step in response to an external call, graphically depicted in Figure 1. The call comes with an object *CEnab* containing some controlled transitions accompanied by bindings for their controlled variables, specifying which controlled transitions are allowed to fire and which tokens can be assigned to their controlled variables. The execution step consists of two phases: first, all the waiting times of tokens are decremented by one (modeling that a simulated time unit has elapsed); second, transitions fire exhaustively and non-deterministically, in compliance with *CEnab*, thus modifying the marking (which constitutes the internal state of the executor). An object *Fseq* containing the transition firings which have taken place, is returned by the executor as a result of the call.

In addition to the calls to perform execution steps, the executor also accepts and processes external calls requesting information about the current marking (e.g. how

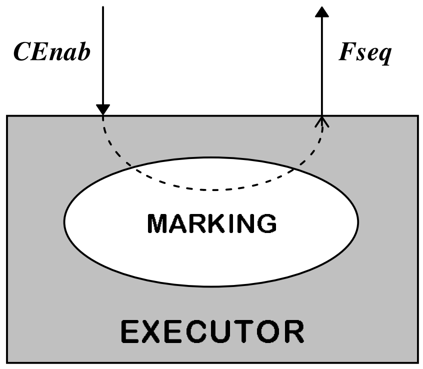

Figure 1: A call to perform an execution step.

many occurrences of a certain token mark a certain place). Of course these other calls do not modify the marking, and many of them may take place between two successive calls to perform execution steps.

In the following subsections, we formalize execution steps by means of suitable mathematical functions.

2.1 Random Stream

Since certain aspects of the working of the executor are non-deterministic, we preliminarily introduce the concept of random stream, which is substantially an infinite sequence of uniformly random values[1]. In following subsections we use random streams as additional arguments to functions, in order to formalize non-deterministic behaviors by means of (deterministic) mathematical functions.

Definition 1. A *random stream* is a family

$$\{z_i\}_{i \in \mathbf{N}}$$

of real numbers such that

$$\forall\, i \in \mathbf{N} : 0 \le z_i < 1.$$

Z is the set of all random streams.

2.2 Decrementing Waiting Times

As mentioned above, the first phase of an execution step consists in decrementing all the waiting times by one (except for null waiting times, which retain their value). This is formalized by the following function.

Definition 2. Given an ESCP-net ***ESCPN***, the function

$$tick \in [M \to M]$$

is defined as follows.

Let $\mu \in M$. For all $p \in P$, we define:

(1) $\forall\, k{:}0 \in TK_{\psi(p)} : tick(\mu)(p)(k{:}0) = \mu(p)(k{:}0) + \mu(p)(k{:}1)$;

(2) $\forall\, k{:}\theta \in TK_{\psi(p)},\ \theta \ge 1 : tick(\mu)(p)(k{:}\theta) = \mu(p)(k{:}\theta{+}1)$.

[1] More precisely, a random stream is just an infinite sequence of values, which we assume randomly generated with uniform probability density.

2.3 Firing a Transition

The firing of a transition with a binding is formalized by the following function.

Definition 3. Given an ESCP-net ***ESCPN***, the function

$$fire \in [D \to M],$$

where

$$D = \{ \langle \mu, t, \beta, \{z_i\}_{i\in\mathbf{N}}\rangle \mid \begin{array}{l} \mu \in M \ \wedge \\ t \in T \ \wedge \\ \beta \text{ binding for } Var(t) \ \wedge \\ t \text{ enabled with } \beta \text{ in } \mu \ \wedge \\ \{z_i\}_{i\in\mathbf{N}} \in Z \ \}, \end{array}$$

is defined as follows.
Let $\langle \mu, t, \beta, \{z_i\}_{i\in\mathbf{N}}\rangle \in D$. Let $a_1, \ldots, a_n \in A$ $(n \geq 0)$ be such that:
(1) $Out(t) = \{a_1, \ldots, a_n\}$;
(2) $a_1 << \ldots << a_n$, where $<<$ is an arbitrary but fixed total order relation over A.
Let

$$\forall j \in \{1,\ldots,n\} : \zeta(a_j) = z_{j-1}.$$

Let μ' be the marking produced by t firing with β and ζ in μ. We define

$$fire(\mu, t, \beta, \{z_i\}_{i\in\mathbf{N}}) = \mu'.$$

The function *fire* returns the marking produced by the firing of t with β in μ, as defined in Definition 16 of [3]. The random generation of the waiting times is obtained from the first n elements $z_0, \ldots, z_{n-1}$ of the random stream $\{z_i\}_{i\in\mathbf{N}}$, which are orderly associated to the outgoing arcs of t according to a total order over the arcs of the ESCP-net.

2.4 Firing a Randomly Chosen Transition

The random choosing of an enabled transition (if any) and its firing, are formalized by the following function.

Definition 4. Given an ESCP-net ***ESCPN***, the function

$$choose\&fire \in [D \to D \cup \{\textbf{exhausted}\}],$$

where

$$D = \{ \langle \mu, CEnab, \{z_i\}_{i\in\mathbf{N}}, Fseq\rangle \mid \begin{array}{l} \mu \in M \ \wedge \\ CEnab \in \mathcal{M}_\omega(\{ \langle t, \beta_C\rangle \mid t \in CT \wedge \beta_C \text{ binding for } \xi(t) \}) \ \wedge \\ \{z_i\}_{i\in\mathbf{N}} \in Z \ \wedge \\ Fseq \in \{ \langle t, \beta\rangle \mid t \in T \wedge \beta \text{ binding for } Var(t) \}^* \ \}, \end{array}$$

is defined as follows.
Let $\langle \mu, CEnab, \{z_i\}_{i\in\mathbf{N}}, Fseq\rangle \in D$. Let

$$Enab = \{ \langle t, \beta\rangle \mid \begin{array}{l} t \in T \ \wedge \\ \beta \text{ binding for } Var(t) \ \wedge \\ t \text{ enabled with } \beta \text{ in } \mu \ \wedge \\ (t \notin CT \ \vee \ \exists\, \beta_C \subseteq \beta : \langle t, \beta_C\rangle \in CEnab) \ \}. \end{array}$$

Since *CEnab* is finite and each place is marked by a finite number of T-tokens, it is easy to see that *Enab* is finite. We define

$$Enab = \emptyset \ \Rightarrow \ choose\&fire(\mu, CEnab, \{z_i\}_{i\in\mathbf{N}}, Fseq) = \textbf{exhausted}.$$

Otherwise, let $\langle t_1, \beta_1\rangle, \ldots, \langle t_n, \beta_n\rangle \in \{ \langle t, \beta\rangle \mid t \in T \wedge \beta$ binding for $Var(t) \}$ $(n \geq 1)$ be such that:

(1) $Enab = \{ \langle t_1, \beta_1\rangle, \ldots, \langle t_n, \beta_n\rangle \}$;

(2) $\langle t_1, \beta_1\rangle << \ldots << \langle t_n, \beta_n\rangle$, where $<<$ is an arbitrary but fixed total order relation over $\{ \langle t, \beta\rangle \mid t \in T \wedge \beta$ binding for $Var(t) \}$.

Let $j \in \{1,\ldots,n\}$ be such that

$$\frac{j-1}{n} \leq z_0 < \frac{j}{n}.$$

Let

$$\mu' = fire(\mu, t_j, \beta_j, \{z_{i+1}\}_{i\in \mathbf{N}}).$$

Let

$$CEnab' = \textbf{if } (\exists\, \beta_C \sqsubseteq \beta_j : \langle t_j, \beta_C\rangle \in CEnab) \textbf{ then } CEnab - \{ \langle t_j, \beta_C\rangle \}_m \textbf{ else } CEnab.$$

Let

$$\{z_i'\}_{i\in\mathbf{N}} = \{z_{i+1+|Out(t_j)|}\}_{i\in\mathbf{N}}.$$

Let

$$Fseq' = Fseq /\!\!/ [\ \langle t_j, \beta_j\rangle\ ].$$

We define

$$choose\&fire(\mu, CEnab, \{z_i\}_{i\in\mathbf{N}}, Fseq) = \langle \mu', CEnab', \{z_i'\}_{i\in\mathbf{N}}, Fseq'\rangle.$$

First, the (finite) set *Enab* of enabled transitions (with relative bindings) in μ is computed, in compliance with *CEnab*. "Compliance" means that only controlled transitions present in *CEnab* may be present in *Enab*, and their bindings in *Enab* must extend (without overwriting) the corresponding bindings in *CEnab*. In case *Enab* is empty, *choose&fire* returns ***exhausted***, indicating that no transition can fire. If, otherwise, *Enab* is non-empty, a pair $\langle t, \beta\rangle$ is randomly chosen from *Enab*, according to the first element z_0 of the random stream. This is achieved by orderly associating the elements of *Enab* to equal-length intervals partitioning the interval between 0 (inclusive) and 1 (exclusive), and then choosing the element of *Enab* associated to the interval where z_0 falls. The function *fire* is then invoked, obtaining a new marking μ' produced by t firing with β in μ (the waiting times for the tokens are generated according to the elements $z_1, \ldots, z_{|Out(t_j)|}$ of the random stream). At this point, if t is controlled, an occurrence of the pair $\langle t, \beta_C\rangle$, where β_C is the restriction of β to the controlled variables of t, is removed from *CEnab* (which must contain such pair because of the way *Enab* is computed), obtaining *CEnab'*. If t is not controlled, *CEnab'* coincides with *CEnab*. Finally, the pair $\langle t, \beta\rangle$ is appended to *Fseq*, obtaining *Fseq'*, and the first $|Out(t)| + 1$ elements of the random stream are eliminated, obtaining $\{z_i'\}_{i\in\mathbf{N}}$. The function *choose&fire* thus returns μ', *CEnab'*, *Fseq'* and $\{z_i'\}_{i\in\mathbf{N}}$ as results.

The reason why not only μ', but also *CEnab'*, *Fseq'*, and $\{z_i'\}_{i\in\mathbf{N}}$ are returned by *choose&fire*, is that, as we will see in the next subsection, *choose&fire* is repeatedly invoked upon the results returned by each previous invocation.

2.5 Exhaustively Firing Randomly Chosen Transitions

As mentioned above, the second phase of an execution step consists in exhaustively and non-deterministically firing transitions, in compliance with *CEnab*. This is formalized by the following function.

Definition 5. Given an ESCP-net ***ESCPN***, the function

$$choose\&fire^* \in [D \rightarrow_p D'],$$

where

$$\begin{aligned} D = \{\langle \mu, CEnab, \{z_i\}_{i\in\mathbf{N}}, Fseq\rangle \mid\ & \mu \in M \ \wedge \\ & CEnab \in \mathcal{M}_\omega(\{ \langle t, \beta_C\rangle \mid t \in CT \wedge \beta_C \text{ binding for } \xi(t) \}) \ \wedge \\ & \{z_i\}_{i\in\mathbf{N}} \in Z \ \wedge \\ & Fseq \in \{ \langle t, \beta\rangle \mid t \in T \wedge \beta \text{ binding for } Var(t) \}^* \ \} \end{aligned}$$

and

$$\begin{aligned} D' = \{ \langle \mu, Fseq\rangle \mid \mu \in M \ \wedge \\ & Fseq \in \{ \langle t, \beta\rangle \mid t \in T \wedge \beta \text{ binding for } Var(t) \}^* \}, \end{aligned}$$

is defined as follows.
Let $\langle \mu, CEnab, \{z_i\}_{i\in\mathbf{N}}, Fseq\rangle \in D$. We define

```
choose&fire*(μ, CEnab, {z_i}_{i∈N}, Fseq) =
    if choose&fire(μ, CEnab, {z_i}_{i∈N}, Fseq) = exhausted
    then ⟨μ, Fseq⟩
    else choose&fire*(choose&fire(μ, CEnab, {z_i}_{i∈N}, Fseq)).
```

The application of *choose&fire** amounts to the repeated application of *choose&fire*, until it returns ***exhausted*** (which indicates that no more transitions are enabled). Through these repeated applications, pair occurrences are removed from *CEnab* as controlled transitions fire, and pairs are appended to *Fseq* as transitions (controlled or not) fire.

Note that the recursive definition of *choose&fire** implies that it is a partial function. The values outside the domain are those for which the iterated application of *choose&fire* does not terminate and goes on forever.

2.6 Performing an Execution Step

An execution step consists in first decrementing waiting times, then exhaustively firing randomly chosen transitions, as formalized by the following function.

Definition 6. Given an ESCP-net ***ESCPN***, the function

$$step \in [D \rightarrow_p D'],$$

where

$$\begin{aligned} D = \{ \langle \mu, CEnab, \{z_i\}_{i\in\mathbf{N}}\rangle \mid \mu \in M \ \wedge \\ & CEnab \in \mathcal{M}_\omega(\{ \langle t, \beta_C\rangle \mid t \in CT \wedge \beta_C \text{ binding for } \xi(t) \} \ \wedge \\ & \{z_i\}_{i\in\mathbf{N}} \in Z \} \end{aligned}$$

and

$$\begin{aligned} D' = \{ \langle \mu, Fseq\rangle \mid \mu \in M \ \wedge \\ & Fseq \in \{ \langle t, \beta\rangle \mid t \in T \wedge \beta \text{ binding for } Var(t) \}^* \}, \end{aligned}$$

is defined as follows.
Let $\langle \mu, CEnab, \{z_i\}_{i\in\mathbf{N}}\rangle \in D$. We define

$$step(\mu, CEnab, \{z_i\}_{i\in\mathbf{N}}) = choose\&fire^*(tick(\mu), CEnab, \{z_i\}_{i\in\mathbf{N}}, []).$$

The arguments of *step* are the current marking μ, a multiset *CEnab* (from an external supervisor), and a random stream $\{z_i\}_{i\in\mathbf{N}}$. First, waiting times are decremented by invoking *tick*, obtaining a marking μ'. Then, *choose&fire** is invoked upon μ', *CEnab*, the random stream, and the empty sequence, obtaining another marking μ'' and a firing sequence *Fseq*, which are returned by *step* as results. μ'' is produced by exhaustively firing transitions in compliance with *CEnab*. *CEnab* is a multiset, instead of simply a set, to allow a same transition to fire more than once with a same binding for its controlled variables. Note that some controlled transitions in *CEnab* may fire less times (possibly none) than their occurrences in *CEnab*. *Fseq* collects all the transitions (and relative bindings) which fire during the execution step, in the order they fire.

Note that, by virtue of the definition above, *step* is partial because *choose&fire** is partial. μ, *CEnab* and $\{z_i\}_{i\in\mathrm{N}}$ are outside the domain of *step* iff they cause non-termination of *choose&fire**. In fact the executor may not terminate an execution step, but this means that the ESCP-net is ill-designed.

3. Implementation Issues

Of course, the specification in the previous section just states *what* the executor does, not *how* it should work. In fact, an actual implementation should use more efficient algorithms than those of the mathematical functions we have defined above. For instance, instead of re-computing the set *Enab* from scratch each time a transition fires, *Enab* should be encoded by means of incrementally modified data structures.

Concerning *Enab*, efficiency dictates the following slight discrepancy between our specification and an actual executor. Conceptually, computing *Enab* requires iterating through all transitions, and for each of them, computing the bindings with which it is enabled (the bindings are computed from scratch if the transition is not controlled, otherwise they are computed starting from those in *CEnab*). Such bindings are found by matching incoming arc expressions with tokens marking input places, and checking guard satisfaction. In addition, it would be necessary to check that no outgoing arc expression or output place stochastic time evaluates to E (otherwise the transition is not enabled with the binding, according to Definition 16 of [3]). However, this further check is not done in an actual executor for efficiency reasons, so that some transitions in *Enab* might really be not enabled. In case one of these transitions is chosen and tentatively fired, the executor should stop with an error. Nevertheless, the occurrence of such a case means that the ESCP-net is ill-designed. In fact, in well-designed ESCP-nets the set *Enab* as computed by an actual executor always contains transitions which are really enabled.

We conclude with a remark about the total order relations among the arcs of an ESCP-nets, and among the pairs $\langle t, \beta\rangle$ of transitions and relative bindings, respectively required by Definition 3 and Definition 4. In an actual executor, there is usually no need of explicitly implementing order relations, as they can be implicitly determined by the inherent order of data structures such as arrays and lists, by which both the set of surrounding arcs of a transition and the set *Enab* are likely to be encoded.

Appendix

In Section 2 we use the mathematical concepts and notations explained below, besides those explained in the Appendix of [3].

If A is a finite set, $|A|$ is its cardinality, i.e. the number of its elements.

If A is a set, $ms \in \mathcal{M}_\omega(A)$, and $a \in A$, we write $a \in ms$ to express that $ms(a) \geq 1$.

If A and B are sets, $[A \rightarrow_{\mathrm{p}} B]$ is the set of all partial functions with domain A and codomain B.

If $f \in [A \rightarrow B]$ and $f' \in [A' \rightarrow B']$, we write $f \subseteq f'$ to express that $A \subseteq A'$ and $f(a) = f'(a)$ for each $a \in A$.

If A is a set, A^* is the set of all finite sequences $[a_1, \ldots, a_n]$ with $n \geq 0$ and $a_1, \ldots, a_n \in A$. [] is the empty sequence. The concatenation $[a_1', \ldots, a_n', a_1'', \ldots, a_m'']$ of two sequences $[a_1', \ldots, a_n']$ and $[a_1'', \ldots, a_m'']$ is denoted by $[a_1', \ldots, a_n'] /\!\!/ [a_1'', \ldots, a_m'']$.

References

[1] A. Camurri, A. Coglio, "A Petri Net-based Architecture for Plant Simulation", in *Proceedings of the 6th IEEE Conference on Emerging Technologies and Factory Automation*, UCLA, Los Angeles, California (USA), September 1997.

[2] A. Camurri, A. Coglio, "Extended Simple Colored Petri Nets: A Tool for Plant Simulation", in *Proceedings of the 1997 IEEE International Conference on Systems, Man, and Cybernetics*, Hyatt Orlando, Orlando, Florida (USA), October 1997.

[3] A. Camurri, A. Coglio, *Extended Simple Colored Petri Nets*, Technical Report, DIST, University of Genoa, Italy, December 1997, available at `ftp://ftp.dist.unige.it/pub/infomus/Publications/escpnets.ps.zip`.