A MARKOV PROCESS FOR REFLECTIVE PETRI NETS

Lorenzo Capra

D.I.Co, Università degli Studi di Milano, Italy

capra@dico.unimi.it

ABSTRACT

The design of dynamic (adaptable) discrete-event systems calls for adequate modeling formalisms and tools able to manage possible changes occurring during system's lifecycle. A common approach is to pollute design with details that do not regard the current system behavior, rather its evolution. That hampers analysis, reuse and maintenance in general. A Petri net based reflective model (based on classical PN) was recently proposed to support dynamic discrete-event system's design, and was applied to dynamic workflow's management. Behind there is the idea that keeping functional aspects separated from evolutionary ones, and applying evolution to the (current) system only when necessary, results in a simple formal model on which the ability of verifying properties typical of Petri nets is preserved. On the perspective of implementing in the short time a discrete-event simulation engine, reflective Petri nets are provided in this paper with a timed state-transition graph semantics, defined in terms of a Markov process.

Keywords: stochastic Petri nets, dynamic systems, evolution, state-transition graph, symbolic techniques.

1. INTRODUCTION

Most existing discrete-event systems are subject to evolution during their lifecycle. Think e.g. of mobile ad-hoc networks, adaptable software, business processes, and so on. Designing dynamic/adaptable discrete-event systems calls for adequate modeling formalisms and tools. Unfortunately, the known well-established formalisms for discrete-event systems, such as classical Petri nets, lack features for naturally expressing possible run-time changes to system's structure. An approach commonly followed consists of polluting system's functional aspects with details concerning evolution. That practice hampers system analysis, reuse and maintenance

A Petri net-based reflective model (Capra and Cazzola 2007) was recently proposed to support dynamic discrete-event system's design, and was successfully applied to specify dynamic workflows (Capra 2008). This approach is based on a reflective layout formed by two logical levels. The achieved clean separation between functional and evolutionary concerns results in a simple formal model for systems exhibiting a high dynamism, in which the analysis capabilities of traditional Petri nets should be preserved.

On the perspective of implementing in the short time a discrete-event simulation engine, the Petri net-based reflective model is provided in this paper with a timed semantics, defined in terms of a Markov process. A crucial issue that is handled is about recognizing possible equivalent base-level's evolutions during simulation. That major topic is managed by exploiting the symbolic state (marking) definition that the particular high-level Petri net flavor used at the meta-level (Stochastic Well formed colored Nets, or SWN) is provided with.

The paper balance is as follows: in section 2 a snapshot of reflective Petri nets is given; in section 3 the main features of the employed Petri net classes are presented; in section 4 a stochastic state-transition graph semantics for reflective Petri nets is defined. Finally section 5 is about work-in-progress. Assuming the readers have some basic knowledge about Petri nets, a semi-formal presentation is adopted, where unessential notions are skipped and simple running examples are used.

2. REFLECTIVE PETRI NETS

The *reflective Petri net* approach permits developers to model a discrete-event system and separately its possible evolutions, and to dynamically adapt system's model when evolution occurs.

The approach is based on a reflective architecture structured in two logical layers (figure 1). The first one, called *base-level*, is an *ordinary Petri net* (a P/T net with priorities and inhibitor arcs) representing the system prone to evolve (*base-level PN*); while the second layer, called *meta-level*, consists of a *high-level Petri net* (a colored Petri net) representing the evolutionary strategies (the meta-program, following the reflection parlance) that drive the evolution of the base-level when certain conditions/events occur.

Meta-level computations in fact operate on a representative of the base-level, called (base-level) *reification*. The reification is defined as a (high-level Petri net) marking, whose a portion, encoding the base-level PN current state (marking), is updated every time the base level Petri net enters a new state. The reification is used by the meta-program to observe (*introspection*) and manipulate (*intercession*) the base-level PN. Any change to the reification is reflected on the base-level PN at the end of a meta-computation (*shift-down*).

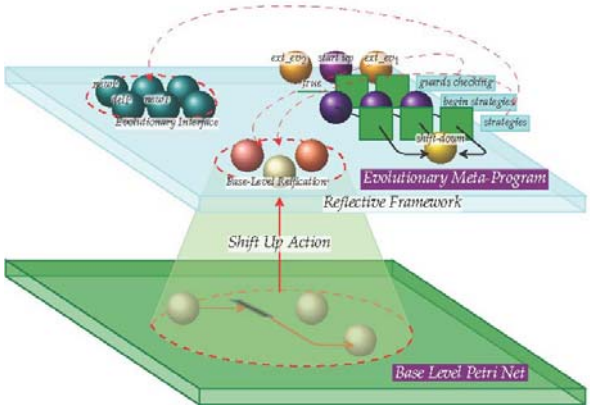

Figure 1: Snapshot of the reflective PN model.

The meta-program is implicitly activated (*shift-up*), then a suitable strategy is put into action, under two conditions: i) either when it is triggered by an external event, or ii) when the base-level enters a given state. The *reflective framework*, a high-level Petri net component as well, is responsible for really carrying out the base-level evolution in a transparent way. Intercession on the base-level PN is carried out in terms of a minimal but complete set of basic operations (called the *evolutionary interface*): addition/removal of places, transitions, arcs - change of transition priorities (base-level's structure change), free moving tokens overall the base-level PN places (base-level's state change).

If one such operation reveals inconsistent, the meta-program is restarted and any changes caused in the meanwhile to the base-level reification are discarded. In other words, evolutionary strategies have a transactional semantics. After a strategy's succeeding execution, changes are reflected down to the base-level Petri net.

Developers have been provided with a tiny ad-hoc language, inspired to Hoare's CSP, that allows anybody to specify his own strategy in a simple way, without any skills in high-level Petri net modeling being required. An automatic translation to a corresponding high-level Petri net is done. Several strategies could be candidate for execution at a given instant: different policies might be adopted in that case to select one, ranging from a deterministic choice to a static assignment of priorities.

According to the reflective paradigm, the base-level runs irrespective of the meta-program, being even not aware of its existence. That raises consistency issues, that are faced by determining, for any strategies, local influence areas on the base-level that are temporarily locked by the meta-level while strategies are being executed.

The interaction between base- and meta- levels, and between meta-level entities, is formalized in (Capra and Cazzola 2007). Let us only outline some essential points:

- The structure of the reflective framework is fixed, while the evolutionary strategies are coupled to the base-level PN, so they vary from time to time. More precisely, the meta-program's model is built according to a predefined pattern, whose the strategies represent the variable component.
- The reflective framework and the meta-program are separated (high-level Petri net) components, sharing two disjoint sets of boundary places denoted hereafter *reification-set* and *evolutionary interface*, respectively. Their composition through a simple place superposition gives rise to the meta-model, called hereafter *meta-level PN*.
- The reification-set is formed by the following colored places: $\{reif_N, reif_M, reif_A, reif_{IT}\}$. The corresponding color domains will be specified later. A well-defined marking of this set of places, hereafter simply denoted *reification*, encodes the structure (including nodes, i.e., places and transitions, arc connections and transition priorities), and the current marking, of the base-level PN. What is most important, there is a one-to-one correspondence, formalized by a bijection, between reifications and P/T nets.
- The initial reification, from time to time, refers to the base-level Petri net modeling the initial system configuration.
- The shift-up, and the reification update following any base-level change of state, are implemented in transparent way at net level, by suitably connecting any base-level PN transition to place $reif_M$, that holds the reification of base-level PN's current marking. The resulting whole model will be hereafter denoted *base-meta PN*.
- The shift-down, i.e., the reflection of changes performed by the meta-program, is modeled by a homonym highest-priority transition of the meta-level PN; it is a kind of meta-transition, that adheres to the usual firing rule as concerns the meta-level PN, further, it makes the (current) base-level PN to be replaced by the P/T net encoded by the reification.

The fixed part of the reflective architecture (the reflective framework) is used to put evolution into practice for any kind of system being modeled. It is responsible for the reflective behavior of the architecture, hiding the work of the evolutionary component to the base-level PN. This approach permits a clean separation between evolution and evolving system, and prevents the base-level PN from being polluted by details related to evolution.

3. SWN BASICS

For (performance) analysis purposes, we decided to use for the base- and meta- levels Generalized Stochastic

Petri nets (GSPN) (Marsan, Balbo, and Conte 1983) and their high-level counterpart, i.e., Stochastic Well formed nets (SWN) (Chiola, Dutheillet, Franceschinis, and Haddad 1993), respectively. This choice has revealed convenient for two reasons: first, the timing semantics of reflective Petri nets is in large part inherited from GSPN (SWN) timing semantics; secondly, the symbolic state representation the SWN formalism is provided with can be exploited to efficiently handle the issues related to recognizing equivalences during model's evolution, as explained in section 4.1.

Let us just recall the basic aspects about SWN (GSPN) timed semantics.

In GSPN (SWN) A priority level is associated to each transition: priority level 0 is reserved for *timed transitions*, while greater priority levels are for *immediate transitions*, which fire in zero time. A *rate*, characterizing an exponential firing delay, is associated to each timed transition, while a *weight* is associated to each immediate transition, to probabilistically solve conflicts between enabled immediate transitions with equal priority.

As a result of this time representation, the *reduced reachability graph* of a GSPN (SWN), i.e., the state-transition graph obtained by suitably removing those markings (called *vanishing*) enabling some immediate transitions, is isomorphic to a Continuous Time Markov Chain (CTMC). Because of the structured syntax of SWN color annotations, behavioral symmetries can be automatically discovered and exploited to build an aggregate state space (called *symbolic reachability graph* or SRG) and a corresponding *lumped CTMC* from a SWN model, according to the *strong lumpability* notion.

As concerns the reflective PN model, while the evolutionary framework, that should be considered as a transparent layer, is formed by immediate transitions only, the evolutionary strategies and, of course, the base-level PN, may also contain timed transitions representing time consuming activities (think e.g. of a network reconfiguration).

3.1. Color Annotations and Symbolic Markings

In Colored Petri nets places, as well as transitions, are associated to *color domains*, i.e., tokens in places have an identifier (color), similarly transitions are parameterized, so that different *color instances* of a given transition can be considered. A *marking m* maps each place p to a multiset on the corresponding color domain, $C(p)$. Any arc connecting p to a transition t is labeled by a function mapping any element of $C(t)$ (i.e., any color instance of t) to a multiset on $C(p)$.

The peculiar and interesting feature of the SWN formalism is the ability of capturing system's symmetries thanks to the structured syntax of color annotations. Efficient analysis/simulation algorithms can be applied that exploit such symmetries. These algorithms rely upon the notion of *symbolic marking* (SM).

SWN color domains are defined as Cartesian products of *basic color classes* C_i, that may be in turn partitioned into *static subclasses* $C_{i,k}$. A SM provides a syntactical equivalence relation on ordinary colored markings: two markings belong to the same SM if and only if they can be obtained from one another by means of permutations on color classes that preserve static subclasses. A SM is formally expressed in terms of dynamic subclasses.

3.1.1. SM formal definition.

The definition of a SM (denoted *sm*) comprises two parts specifying the so called dynamic subclasses and the distribution of colored symbolic tokens (tuples built of dynamic subclasses) over the net places, respectively.

Dynamic subclasses define a parametric partition of color classes preserving static subclasses: let D_i and s_i denote the set of dynamic subclass of C_i (in *sm*), and the number of static subclasses of C_i (if C_i is not split then $s_i = 1$). The j-th dynamic subclass of C_i, $Z_j^i \in D_i$, refers to a static subclass, denoted $d(Z_j^i)$, $1 \le d(Z_j^i) \le s_i$, and has an associated cardinality $|Z_j^i|$, i.e., it represents a parametric set of colors (in the sequel we shall consider cardinality one dynamic subclasses). It must hold, for each k : 1...s_i

$$\sum_{j:d(Z_j^i)=k} |Z_j^i| = |C_{i,k}| \quad (1)$$

The token distribution in *sm* is defined by a function (denoted itself *sm*) mapping each place p to a multiset on the *symbolic color domain* of p, obtained replacing each C_i with D_i in $C(p)$.

Among several possible equivalent representations, the canonical representative provides SM with an univocal formal expression, based on a lexicographic ordering of dynamic subclass distribution over the net places.

4. A STATE-TRANSITION SEMANTICS FOR REFLECTIVE PN

On the light of what said in section 2, the behavior of a reflective PN model between any meta-level activation and the consequent shift-down is naturally described in terms of (stochastic) Petri net state-transitions.

A state m_i is simply an ordinary marking of the base-meta PN, the Petri net obtained by suitably composing the base-level PN (a GSPN) and the meta-level PN (a SWN). Then, letting t ($t \neq$ *shift-down*) be any transition enabled in m_i, according to the GSPN (SWN) firing rules, and m_j be the marking reached upon its firing, we have the labeled state-transition:

$$m_i \xrightarrow{\lambda(t)} m_j \quad (2)$$

where $\lambda(t)$ denotes the weight, or the exponential rate, associated with t, depending on whether t is timed or immediate.

There is nothing else to do but consider the case where m_s is a vanishing marking enabling the higher-priority pseudo-transition *shift-down*: then

$$m_s \xrightarrow{w=1} m'_o \quad (3)$$

m'_0 being the marking of the new base-meta PN, obtained first by replacing the (current) base-level PN with the GSPN isomorphic to the reification marking (once it has been suitably connected to the meta-level PN), then by firing *shift-down* as it were a normal immediate transition.

Using the same technique for eliminating vanishing states as in the reduced reachability graph algorithm for GSPN (SWN), it is possible to build a CTMC from the labeled state-transition graph of the reflective PN model.

4.1. Recognizing equivalent base-level evolutions

The state-transition graph semantics just introduced precisely defines the (timed) behavior of a reflective Petri net model, but suffers from two evident drawbacks. First, it is highly inefficient: the state description is exceedingly redundant, comprising a large part concerning the evolutionary strategy, which is unnecessary to describe the evolving system.

The second concern is even more critical, and indirectly affects efficiency: there is no way of recognizing whether the system, during its dynamics/evolution, reaches equivalent configurations. Deciding about system's state-transition graph finiteness and ergodicity are major performance analysis issues that are strictly related to the ability of recognizing equivalent behaviors/evolutions of the modeled system. More generally, a number of techniques based on state-space inspection rely on this ability.

For example, it may happen that (apparently) different strategies cause in truth equivalent structural changes to the base-level Petri net (the evolving system), that cannot be identified by the definition of state provided before. The combined effect of different sequences of evolutionary strategies might produce the same effects. Even more likely, the internal dynamics of the evolving system might lead to reach equivalent configurations.

The above tricky question, that falls into a graph isomorphism sub-problem, as well as the global efficiency of the approach, are tackled by resorting to the peculiar characteristic of SWN: the symbolic marking notion. The color domains of the meta-level PN are built of color class *Node*, representing the base-level PN nodes (places plus transitions) at the meta-level. As concerns the reification-set, they are:

$$\begin{aligned} &C(reif_N), C(reif_M), C(reif_\Pi) : Node \\ &C(reif_A) : Node \times Node \end{aligned} \quad (4)$$

Class *Node* is logically partitioned into places and transitions. More precisely, *Node* is defined as:

$$\underbrace{\overbrace{p_1 \cup \ldots p_k}^{named_P} \bigcup Unnamed_P}_{\text{places}} \bigcup \underbrace{\overbrace{t_1 \cup \ldots t_m}^{named_T} \bigcup Unnamed_T}_{\text{transitions}} \quad (5)$$

.

Symbols $\{p_i\}$, $\{t_j\}$ denote singleton static subclasses. Conversely, subclasses $Unnamed_P$ and $Unnamed_T$ should be normally large enough to be considered as logically unbounded repositories of anonymous places/transitions.

Behind there is a simple intuition: while some ("named") nodes, for the particular role they play, preserve their identity during base-level's evolution, and may be explicitly referred to during base-level's manipulation, others ("unnamed") are undistinguishable from one another. In other words any pair of "unnamed" places (transitions) might be freely exchanged on the base-level PN, without altering the model's semantics.

There are two extreme cases: $named_P$ ($named_T$) = Ø, and, on the opposite, $Unnamed_P$ ($Unnamed_T$) = Ø. The former meaning that all places/transitions can be permuted, the latter instead that all nodes are distinct.

The technique we use to recognize equivalent base-level evolutions relies on the base-level reification and the adoption of a symbolic state representation for the base-meta PN that, we recall, results from composing in transparent way the base-level PN and the meta-level PN.

First we have to set as *initial state* of the reflective PN model a symbolic marking (sm_0) of the base-meta PN instead of an ordinary one: any dynamic subclass of $Unnamed_P$ ($Unnamed_T$) will represent an arbitrary "unnamed" place (transition) of the base-level PN.

Because of the simultaneous update mechanism of the reification (section 2), and the consequent one-to-one correspondence between the current base-level PN and the reification at the meta-level (Capra and Cazzola 2007), we can state the following

Definition 1 (equivalence relation). Let sm_i, sm_j be two states of the reflective Petri net model.
$sm_i \equiv sm_j$ if and only if their projections on the reification-set have the same canonical representative.

Consider the very simple example in figure 2, that depicts three base-level PN configurations, at different time instants. The convention we adopt is that while symbols t_2 denotes a "named" transition, symbols x_i and y_j denote "unnamed" places and transitions, respectively. In other words (abusing notation) x_i and y_j will denote also dynamic subclasses of $Unnamed_P$ and $Unnamed_T$, respectively. We assume that all transitions have the same priority level.

We can observe that the base-level PNs on the top and on the middle have the same structure, but (apparently) different current marking. We can imagine that they represent a possible (internal) dynamics of the base-level Petri net.

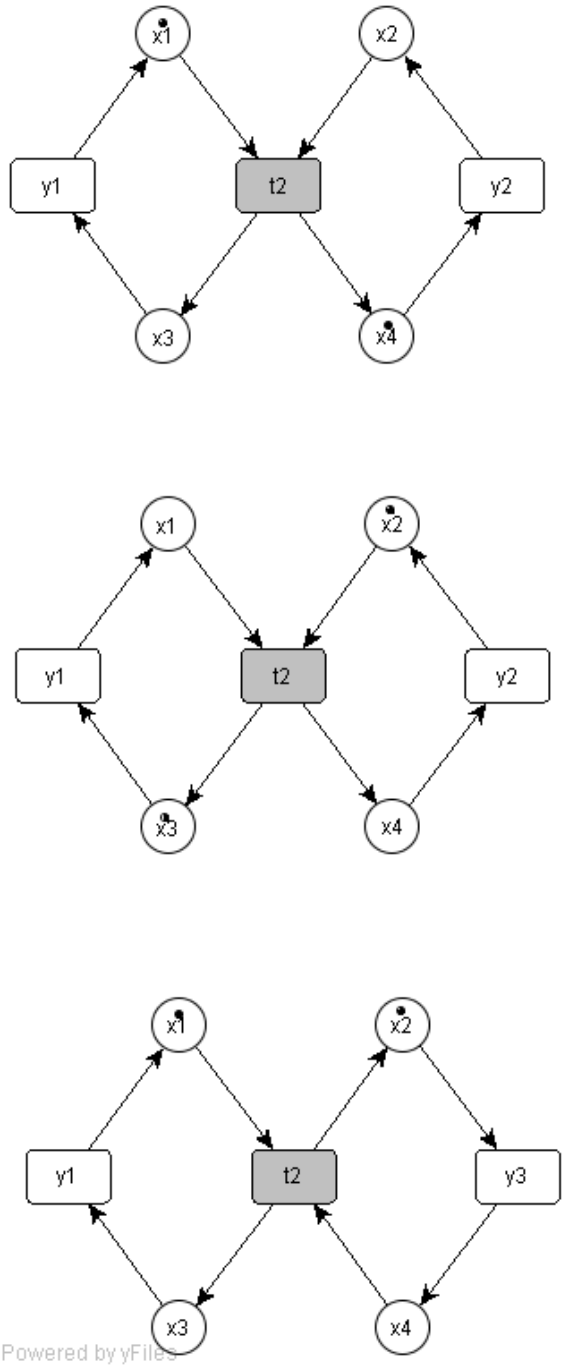

Figure 2: Three equivalent base-level PN

Conversely, we might think of the Petri net on the bottom of figure 2 as an (apparent) evolution of the base-level PN on the top, in which transition y_2 has been replaced by a new transition (y_3), new connections are set, and a new marking is defined.

Nevertheless, the three base-level configurations are equivalent, according to definition 1. It is sufficient to take a look at their respective reifications, that are encoded as symbolic markings (hereafter multisets are expressed as formal sums).

Consider first the base-level PNs on the top and on the middle of figure 2, whose reification are:

$$sm(reif_N) = y_1 + y_2 + t_2 + x_1 + x_2 + x_3 + x_4$$
$$sm(reif_M) = x_1 + x_4$$
$$sm(reif_A) = < x_1, t_2 >+< t_2, x_3 >+< x_3, y_1 >+< y_1, x_1 >+ < x_2, t_2 >+ < t_2, x_4 > + < x_4, y_2 > + < y_2, x_2 > \quad (6)$$

and

$$sm'(reif_N) = y_1 + y_2 + t_2 + x_1 + x_2 + x_3 + x_4$$
$$sm'(reif_M) = x_3 + x_2$$
$$sm'(reif_A) = < x_1, t_2 >+< t_2, x_3 >+< x_3, y_1 >+< y_1, x_1 >+ < x_2, t_2 >+ < t_2, x_4 > + < x_4, y_2 > + < y_2, x_2 > \quad (7)$$

respectively.

They can be obtained from one another by the following permutation of "unnamed" places and transitions (we denote by $a \leftrightarrow b$ the bidirectional mapping: $a \rightarrow b$, $b \rightarrow a$):

$$\{x_1 \leftrightarrow x_2, x_3 \leftrightarrow x_4, y_1 \leftrightarrow y_2\} \quad (8)$$

hence, they are equivalent.

With similar arguments we can show that the base-level PN on the top and on the bottom of figure 2 are equivalent too. The bottom's Petri net reification is:

$$sm''(reif_N) = y_1 + y_3 + t_2 + x_1 + x_2 + x_3 + x_4$$
$$sm''(reif_M) = x_1 + x_2$$
$$sm''(reif_A) = < x_1, t_2 >+< t_2, x_3 >+< x_3, y_1 >+< y_1, x_1 >+ < x_2, y_3 >+ < y_3, x_4 > + < x_4, t_2 > + < t_2, x_2 > \quad (9)$$

sm and *sm″* can be in turn obtained from one another by the following permutation:

$$\{x_2 \leftrightarrow x_4, y_3 \leftrightarrow y_2\} \quad (10)$$

The canonical representative for these three equivalent base-level PN's reifications (i.e., states of the reflective PN-model), computed according to the corresponding SWN algorithm, turns out to be *sm*.

5. CONCLUSIONS AND FUTURE WORK

We have semi-formally presented a (timed) state-transition graph semantics for reflective Petri nets, a formalisms well suited to model evolvable discrete-event systems, based on classical stochastic Petri nets (GSPN, and their high-level version, SWN). In particular, we have addressed major topics related to recognizing equivalent system's evolutions, by exploiting the SWN's symbolic state notion. We are planning to integrate the GreatSPN tool, that natively supports GSPN and SWN, with new modules for the graphical editing of reflective PN models, and their analysis/simulation based on the associated state-transition semantics. We are also investigating possible applications of reflective Petri nets for the analysis of

dynamic workflows (already specified using the same formalism) and protocols for mobile ad-hoc networks.

ACKNOWLEDGMENTS

The author thanks Walter Cazzola, who contributed to the definition of Reflective PN, for its valuable suggestions .

REFERENCES

Capra, L., 2008. Addressing soundness and efficiency issues in dynamic processes: a reflective pn-based modeling approach. *In Proceedings of SCS Spring Simulation Multiconference (SpringSim'08) - Business and Industry Symposium*, April 14 – 17, Ottawa (Canada).

Capra, L., and Cazzola, W., 2007. Self-evolving petri nets. *Journal of Universal Computer Science*, 13(13): pp 2002-2034, available from: http://www.jucs.org/jucs_13_13/self_evolving_petri_nets [accessed 17 Dec 2007]

Chiola, G., Dutheillet, C., Franceschinis, G., and Haddad, S., 1993. Stochastic Well-Formed Coloured Nets for Symmetric Modelling Applications. *IEEE Transactions on Computers*, 42(11):pp 1343–1360.

Chiola, G., Franceschinis, G., Gaeta, R., and Ribaudo, M., 1995. GreatSPN 1.7: Graphical editor and analyzer for timed and stochastic Petri nets. *Performance Evaluation*.

Marsan, M. A., Balbo, G., and Conte, G., 1983. A class of generalised stochastic petri nets for the performance evaluation of multiprocessor systems. *In SIGMETRICS'83: Proceedings of the ACM SIGMETRICS conference on Measurement and modeling of computer systems*, pp 198–199, New York (NY, USA).

AUTHORS BIOGRAPHY

Lorenzo Capra was born in Monza (Italy), and went to the University of Milan, where he obtained his Laurea degree in Computer Science in 1992. After having collaborated for several years with the Automation Research Center at the National Electric Power Provider (ENEL), he moved to the University of Turin, where he received a Ph.D in Computer Science. He is currently assistant professor at the Dept. of Informatics and Communication (Di.C.O) at the University of Milan. His research interests include High-Level Petri Nets analysis/simulation and formal methods in software engineering.