Proceedings of the World Congress on Engineering 2009 Vol I
WCE 2009, July 1 - 3, 2009, London, U.K.

A Symbolic State-Transition Graph for a Class of Dynamic Petri Nets

Lorenzo Capra *

Abstract—**The design of dynamic, adaptable discrete-event systems calls for adequate modeling formalisms and tools in order to manage possible changes occurring during system's lifecycle. A common approach is to pollute the design with details not concerning the current system behavior, rather its evolution. That hampers analysis, reuse and maintenance in general. A Petri net-based reflective model (based on classical Petri nets) 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 clean formal model for dynamic systems. This model preserves the ability of verifying properties typical of classical Petri nets. As a first step toward the implementation (in the short time) of a discrete-event simulator, Reflective Petri nets are provided in this paper with a semantics defined in terms of labeled state-transitions.**

Keywords: 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 [1], 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.

Reflective Petri nets [2] have been recently proposed as design framework for dynamic discrete-event systems, and successfully applied to dynamic workflows [3]. They rely 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, which should preserve the analysis capabilities of classical Petri nets.

On the perspective of implementing in the short time an automatic solver and a discrete-event simulation engine, Reflective Petri nets are provided in this paper with a (labeled) state-transition semantics. Any analysis/simulation techniques based on state-space inspection has to face a crucial question, that is how to recognize possible equivalent states during base-level's evolution. That major topic is managed by exploiting the symbolic state definition the particular Colored Petri net flavor [4] used for the meta-level is provided with, and represents the paper's original contribution.

The balance is as follows: background information on Reflective Petri nets and the employed Petri net formalisms are given in sections 2,3. The focus is put there on those elements directly connected to the paper's main contribution, the definition of a state-transition semantics for Reflective Petri nets (section 4). An application of the semantics to a dynamic system taken from literature is summarized in section 5. Related works are mentioned and discussed in section 6. Finally section 7 is about work-in-progress. Assuming the readers have some basic knowledge about Petri nets, a semi-formal presentation is adopted, and a few simple examples are used.

2 WN Basics

The formalisms employed for the two layers (meta- and base-level) of the reflective layout are Well-formed Nets (WN) [5], a flavor of Colored Petri nets (CPN) [4], and their unfolded counterpart, an extension of ordinary Place/Transition nets [1], respectively. This choice has revealed convenient for two main reasons: first, the behavior of Reflective Petri nets can be formally stated in terms of classical Petri nets; secondly, the symbolic state representation the WN formalism is provided leads to an effective state-transition semantics for Reflective Petri nets, in which the intriguing question related to recognition of equivalent evolutions is efficiently handled.

There is also a third reason, which is interesting in the perspective of doing performance analysis: our feeling is that if we considered the stochastic extension of

*Department of Informatics and Communication (D.I.Co), Università degli Studi di Milano, MI Italy 20139, Phone/Fax +390250316256/373, Email: capra@dico.unimi.it

WNs (SWN) [6], and their unfolding, that is Generalized Stochastic Petri nets (GSPN) [7], for the meta- and base-level, we might set a timing semantics for Reflective Petri nets inherited in large part from the GSPN (SWN) timing semantics (which is defined in terms of a Markov process). In the sequel of this work only the untimed behavior of Reflective Petri nets will be defined.

While retaining CPN's expressive power, WNs are characterized by a structured syntax which is used by efficient analysis algorithms, based on the symbolic marking notion. This section does not present all the features of WNs, for which the reader is suggested to look at the original work, but just introduces them informally, focusing on the symbolic marking definition.

Unlike CPNs, WNs (and their unfolding as well) include transition priorities and inhibitor arcs. Both these features enhance the CPN formalism expressiveness. In particular, priorities are useful to represent the transactional execution of evolutionary strategies, as well as the implicit synchronization between base- and meta-level.

Because of the structured syntax of WN color annotations, behavioral symmetries can be automatically discovered and exploited to build an aggregate state space (called *symbolic reachability graph* or SRG) and (in SWN) a corresponding *lumped CTMC*.

2.1 WN color annotations

In Colored Petri nets [8] 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* $\mathbf{M}$ maps each place p to a multiset on the corresponding color domain $\mathcal{C}(p)$. Any arc connecting p to a transition t is labeled by a function mapping any element of $\mathcal{C}(t)$ (i.e., any color instance of t) to a multiset on $\mathcal{C}(p)$.

The peculiar and interesting feature of the WN 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.

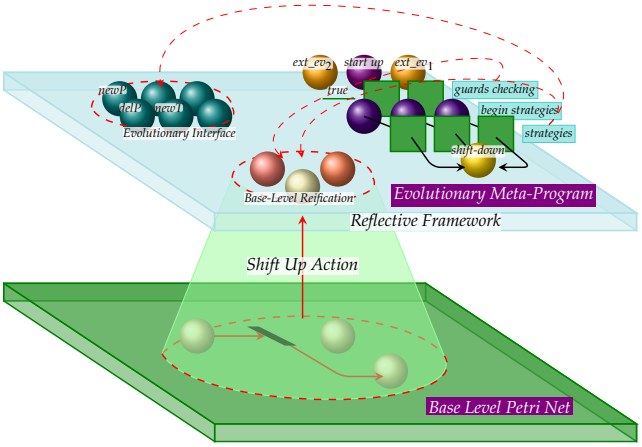

Figure 1: A snapshot of the reflective PN model.

2.2 The Symbolic Marking (SM) notion.

The definition of a SM (denoted $\widehat{\mathbf{M}}$) [9] 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 $\widehat{\mathbf{M}}$), 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 $d(Z_j^i)$, $1 \leq d(Z_j^i) \leq s_i$, 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 $k : 1...s_i$, $\sum_{j:d(Z_j^i)=k} |Z_j^i| = \mid C_{i,k} \mid$

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

Among several possible equivalent representations, the canonical representative [9] provides SM with a univocal formal expression, based on a lexicographic ordering of dynamic subclass distribution over the net places.

3 Reflective Petri Nets

The *reflective Petri nets* 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(s) the evolution of the base-level when certain conditions/events occur.

Meta-level computations operate on a representative of the base-level, called *reification*. It is defined as a (high-level Petri net) marking, a portion of which (encoding the base-level current state) 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. The changes to the reification are reflected to the base-level at the end of a meta-computation (*shift-down*).

The meta-program is implicitly activated (*shift-up*), then a suitable strategy (evolution rule) is put into action, under two conditions: i) either when it is triggered by an external event, or ii) the base-level enters a given state. The meta program can also include an evolution model, that is, an evolutionary strategy which does not depend on the current state of the base-level. That capability enhances the flexibility of the reflective layout.

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. It should be considered as a transparent (meta-)layer of the reflective model, therefore it is formed by higher-priority transitions. 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 meantime to the base-level reification are discarded. Trying to delete a node/arc yet not existing is an example of inconsistent operation. The consistency checks are automatically performed by the framework, the evolutionary engine of the whole layout. In other words, the evolutionary strategies have a transactional semantics. Only after a strategy's succeeding run, 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 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 intriguing consistency issues, which are faced by determining, for any strategies, local influence areas on the base-level. One such area is temporarily locked by the meta-program, until the corresponding strategy is completed.

The interaction between base- and meta- levels, and between meta-level entities, is formalized in [2]. Let us only outline here some essential points:

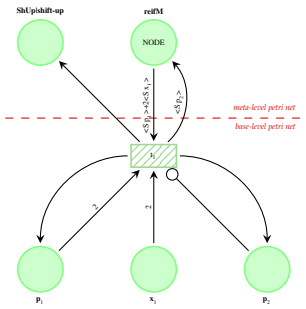

Figure 2: Reification implemented at net level.

1. 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.

2. The reflective framework (the fixed part of the reflective layout) 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*.

3. The reification-set is formed by the following colored places: $\{\texttt{reifN}, \texttt{reifA}, \texttt{reif}\Pi, \texttt{reifM}\}$. The corresponding color domains, which will be specified later, are built of basic color class $Node$, a logically unbounded repository holding all potential base-level nodes, considering any evolutions. A well-defined marking of the set above, hereafter simply denoted *reification*, encodes the structure (i.e., nodes, arc connections and transition priorities, respectively), 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.

4. The initial reification refers to the base-level Petri net modeling the initial system configuration.

5. The shift-up, and the reification update, are implemented in transparent way at net level by suitably connecting every base-level PN transition to place `reifM` , which holds the reification of base-level PN's current marking. The resulting model will be hereafter denoted *base-meta PN*. The idea is illustrated in Fig. 2. Any state changes at the base-level Petri net are instantaneously reproduced on the reification, conceptually maintaining base-level's unawareness about the meta-program. As an example, the firing of transition t_1 results in withdrawing one and two tokens from places p_1 and x_1, respectively, and putting one in p_2. While token consumption is emulated by (input) arc function $\langle S\, p_1\rangle + 2 \cdot \langle S\, x_1\rangle$, token production is emulated by (output) arc function $\langle S\, p_2\rangle$. A complete splitting of class $Node$ is required to refer to any places added from to time to the base-level (e.g., x_1), by means of WN constant functions.

6. The reflection of changes performed by the meta-program, or shift-down, is simulated by the homonym highest-priority transition of the meta-level PN; it is a kind of meta-transition, which 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.

4 State-transition semantics for reflective PN

The usage of the WN formalism [5] at the meta-level of the reflective layout permits a simple, effective characterization of reflective Petri net's behavior, in terms of state-transitions. Using an apparently low-level semantics, as the state-transitions one, rather than an higher-level semantics defined in terms of an expressive standard Petri net class (e.g., algebraic nets) is justified by efficiency needs. In the perspective of implementing a discrete-event simulator for reflective Petri nets, the WN's symbolic state notion can be exploited to identify and fold possible equivalent evolutions of a Reflective Petri net model.

On the light of the causal connection established between base- and meta-level, the behavior of a Reflective Petri net model between consecutive meta-level activations is fully described in terms of a WN model: the meta-level PN, which includes (or better, is suitably connected to) an uncolored part, the base-level PN. This model will be hereafter denoted *base-meta PN*. Therefore, it comes to be natural providing Reflective Petri nets with the notion of state below:

Definition 1 (state) *A state of a Reflective Petri net is a marking* $\mathbf{M}_i$ *of the base-meta PN obtained by suitably composing the base-level PN and the meta-level PN.*

Letting t ($\neq$ `shiftdown`) be any transition (color instance) enabled in $\mathbf{M}_i$, according to the classical Petri net's firing rule, and $\mathbf{M}_j$ be the marking reached upon its firing, we have the labeled state-transition

$$\mathbf{M}_i \quad \stackrel{t}{\longrightarrow} \quad \mathbf{M}_j$$

There is nothing to do but consider the case where m_f is a marking enabling the pseudo-transition *shift-down*: then,

$$\mathbf{M}_f \quad \stackrel{shift-down}{\longrightarrow} \quad \mathbf{M}'_0,$$

$\mathbf{M}'_0$ being the marking of the base-meta PN obtained by first replacing the (current) base-level PN with that one which is isomorphic to the reification marking (once it has been suitably connected to the meta-level PN), then firing *shift-down* as it were an ordinary transition.

4.1 Recognizing Equivalent 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 meta-level PN, 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 modeled system, during its dynamics/evolution, reaches equivalent states.

The ability of deciding about a system's state-transition graph finiteness and strongly-connectedness (both issues being strictly related to the ability of recognizing equivalent states) are in fact mandatory preconditions for performance analysis: we know that a sufficient condition for a finite CTMC to have stationary solution (steady-state) is to include one maximal strongly connected component. More generally, any technique based on state-space inspection relies on the ability above.

Recognizing equivalent evolutions is a tricky question. For example, it may happen that (apparently) different strategies cause in truth equivalent transformations to the base-level Petri net (the evolving system), that cannot be identified by Def. 1. Yet, 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 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 WN: 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:

$$\begin{aligned} \mathcal{C}(reifN), \mathcal{C}(reifM), \mathcal{C}(reif) &: Node \\ \mathcal{C}(reifA) &: Node \times Node \end{aligned}$$

Arcs are represented by 2-tuples belonging to $Node \times Node$, since there are no inhibitor arcs in the running example (Fig. 3).

Class $Node$ is logically partitioned as follows:

$$\underbrace{\overbrace{p_1 \cup \ldots p_k}^{named_p} \cup Unnamed_p}_{places} \cup \underbrace{\overbrace{t_1 \cup \ldots t_n}^{named_t} \cup Unnamed_t}_{transitions}.$$

Symbols $\{p_i\}$, $\{t_j\}$ denote singleton static subclasses. Conversely, $Unnamed_p$ and $Unnamed_t$ are static subclasses collecting all anonymous (i.e., indistinguishable) places/transitions.

The intuition behind is simple: while some ("named") nodes, for the particular role they play, preserve the identity during base-level evolution, and may be explicitly referred to during base-level manipulation, others ("unnamed") are indistinguishable 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: $named_p$ ($named_t$) $= \emptyset$ and, on the opposite, $Unnamed_p$ ($Unnamed_t$) $= \emptyset$. The former meaning that all places/transitions can be permuted, the latter instead that all nodes are distinct. Note that this static partition is different from that one actually used on the base-meta PN, where any places of base-level PN must be explicitly referred to when connecting the base-level PN to the meta-level PN (Fig. 2).

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.

We only have to set as initial state of the Reflective Petri net model a symbolic marking ($\widehat{\mathbf{M}}_0$) 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, and the consequent one-to-one correspondence along the time between the current base-level PN and its reification at the meta-level, we can state the following:

Definition 2 (equivalence relation) *let $\widehat{\mathbf{M}}_i$, $\widehat{\mathbf{M}}_j$ be two symbolic states of the reflective Petri net model. $\widehat{\mathbf{M}}_i \equiv \widehat{\mathbf{M}}_j$ if and only if their restrictions on the reification set of places have the same canonical representative.*

Lemma 1 *let $\widehat{\mathbf{M}}_i \equiv \widehat{\mathbf{M}}_j$. Then the base-level PNs at states $\widehat{\mathbf{M}}_i$ and $\widehat{\mathbf{M}}_j$ are isomorphic.*

Consider the very simple example in Fig. 3, that depicts three base-level PN configurations, at different time instants. The hypothesis is that while symbols t_2 denote a "named" transition, symbols x_i and y_j denote "unnamed" places and transitions, respectively. We assume that all transitions have the same priority level, so we disregard the reification of priorities.

We can observe that the Petri nets on the left and on the middle are the same, but for their current marking: we can imagine that they represent a possible (internal) dynamics of the base-level Petri net, corresponding to the firing sequence $y_3; t_2; y_3$. Conversely, we might think of the right-most Petri net as an (apparent) evolution of the base-level PN on the left, in which new connections are set, and a new marking is defined.

Nevertheless, the three base-level configurations are equivalent, according to definition 2. It is sufficient to take a look at their respective reifications, that are encoded as symbolic markings (multisets are expressed as formal sums): consider for instance the base-level PN on the left and on the middle of Fig. 3, whose reification are:

$\widehat{\mathbf{M}}(\texttt{reifN}) = y_1 + y_3 + t_2 + x_1 + x_2 + x_3 + x_4,$

$\widehat{\mathbf{M}}(\texttt{reifM}) = x_1 + x_4,$

$\widehat{\mathbf{M}}(\texttt{reifA}) = \langle x_1, t_2\rangle + \langle t_2, x_3\rangle + \langle x_3, y_1\rangle + \langle y_1, x_1\rangle + \langle x_2, t_2\rangle + \langle t_2, x_4\rangle + \langle x_4, y_3\rangle + \langle y_3, x_2\rangle$

and

$\widehat{\mathbf{M}}'(\texttt{reifN} = y_1 + y_3 + t_2 + x_1 + x_2 + x_3 + x_4,$

$\widehat{\mathbf{M}}'(\texttt{reifM}) = x_3 + x_2,$

$\widehat{\mathbf{M}}'(\texttt{reifA}) = \langle x_1, t_2\rangle + \langle t_2, x_3\rangle + \langle x_3, y_1\rangle + \langle y_1, x_1\rangle + \langle x_2, t_2\rangle + \langle t_2, x_4\rangle + \langle x_4, y_3\rangle + \langle y_3, x_2\rangle$

respectively. They can be obtained from one another by the permutation ($a \leftrightarrow b$ denotes the bidirectional mapping: $a \to b, b \to a$):

$$\{x_1 \leftrightarrow x_2, x_3 \leftrightarrow x_4, y_1 \leftrightarrow y_3\},$$

in other words, they are equivalent.

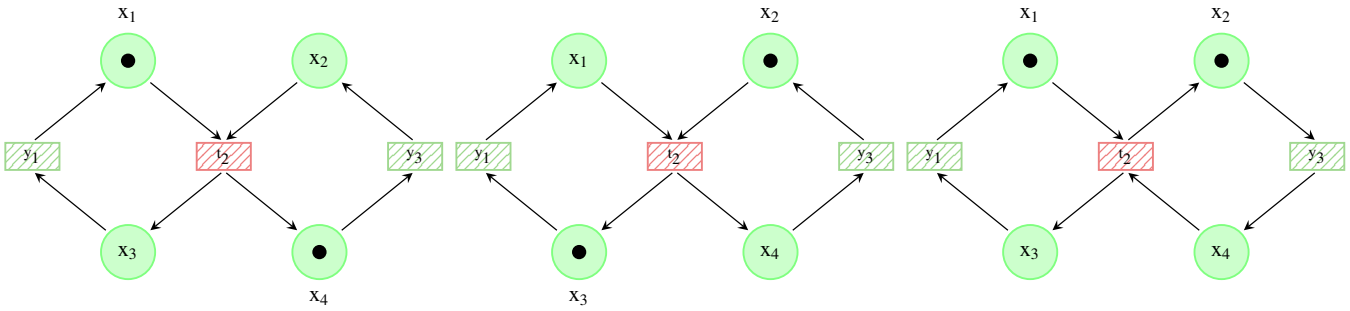

Figure 3: Equivalent base-level Petri Nets

Using similar arguments we could show that the base-level PN on the left and on the right of Fig. 3 are equivalent too. The right-hand Petri net's reification is:

$\widehat{\mathbf{M}}''(\texttt{reifN}) = y_1 + y_3 + t_2 + x_1 + x_2 + x_3 + x_4$

$\widehat{\mathbf{M}}''(\texttt{reifM}) = x_1 + x_2$

$\widehat{\mathbf{M}}''(\texttt{reifA}) = \langle x_1, t_2\rangle + \langle t_2, x_3\rangle + \langle x_3, y_1\rangle + \langle y_1, x_1\rangle + \langle x_2, y_3\rangle + \langle y_3, x_4\rangle + \langle x_4, t_2\rangle + \langle t_2, x_2\rangle$

$\widehat{\mathbf{M}}$ and $\widehat{\mathbf{M}}''$ can be in turn obtained from one another by the permutation: $\{x_2 \leftrightarrow x_4\}$. Their canonical representative is $\widehat{\mathbf{M}}$.

A transition between symbolic states of of the reflective Petri net model is defined as follows.

Definition 3 (symbolic state's transition) *Let σ be a (possibly empty) meta-level PN's transition firing sequence, $\texttt{shiftdown} \notin \sigma$. Then*

$$\widehat{\mathbf{M}}_i \xrightarrow{t} \widehat{\mathbf{M}}_j,$$

if and only if t is either `shiftdown` *or a base-level PN's transition, and there exist $\sigma, \widehat{\mathbf{M}}'_i$*

$$\widehat{\mathbf{M}}_i \xrightarrow{\sigma} \widehat{\mathbf{M}}'_i \xrightarrow{t} \widehat{\mathbf{M}}_j$$

Of course $\widehat{\mathbf{M}}'_i$, as well as any intermediate marking crossed by σ, is equivalent to $\widehat{\mathbf{M}}_i$. This concept is illustrated in Fig. 4. Rounded boxes represent classes of equivalent markings: they are the nodes of the symbolic state-transition graph describing the behavior of a Reflective Petri net. Visible state-transitions are only due to the occurrence of either *shiftdown*, or any base-level PN transition (like t in the picture). Meta-level transition sequences (σ) between equivalent markings (represented by circles) are represented by dashed arrows, meaning that they are not visible by an external observer. If the meta-level PN never enters a deadlock or a livelock, the reachability information provided by the symbolic state-transition graph are complete and reliable.

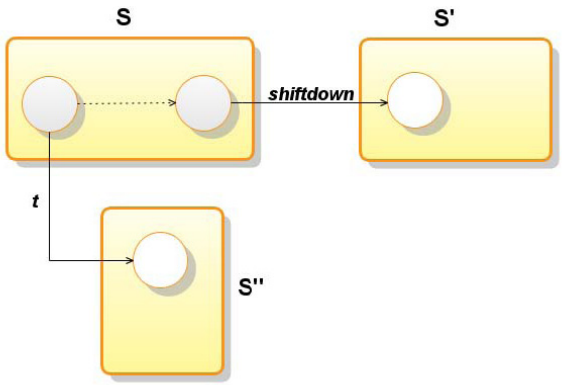

Figure 4: Transitions between state-equivalent classes

5 An application example

The (symbolic) state-transition semantics of Reflective Petri nets has been tested on the *hurried philosophers* problem [10], a variant of the well known dining philosophers problem that introduces a high dynamism and mobility degree. The version here considered meets the following requirements:

- two philosophers initially sit on the table;
- a philosopher can start eating only when he contemporary gets both the adjacent forks, according to the classical problem's formulation;
- a philosopher sitting on the table has additional faculties: he/she can invite a colleague (arbitrarily chosen) to join the table and sit down on either side; he/she can ask one of his/her neighbors to leave the table

each philosopher is going around with his own fork; when he joins the table he keeps the fork, when he leaves he brings it with him.

The base-level Petri net representing the initial system, as well as the meta-program which is in charge of man-

aging the evolution, are specified in [2]. Any invitation/chasing away requests activate the meta-program. The arrival/departure of a philosopher causes a local reconfiguration of the table, in which the fork sharing by adjacent philosophers is rearranged in a consistent way.

Table 5 reports some evidences of the effectiveness of the symbolic state representation adopted for Reflective Petri nets, versus the ordinary one. The experiment was conducted using the GreatSPN tool, and an ad-hoc script procedure emulating the shift-down effect on the base-level. The first column indicates the problem size, i.e., the table capacity. All philosophers can be permuted with one another. We can appreciate a sensible reduction, even for small problem sizes, due to the very high symmetry degree exhibited by the system during its evolution.

Table 1: Symbolic vs. ordinary state-space size

#philosophers	symbolic states	ordinary states
3	423	2567
4	2843	374809
5	23560	1765836
6	147812	96905034
7	960345	n.a.
8	4767385	n.a.
9	12097086	n.a.

6 Related Works

Many efforts have been devoted in trying to extend Petri nets with dynamical features. Follows a non-exhaustive list of relevant proposals appeared in the literature. In [11], the author is proposing his pioneering work, *self-modifying* nets, in which the flow relation between a place and a transition is a linear function of the place marking. Another major contribution of Valk is the so-called *nets-within-nets* paradigm [12], where tokens flowing through a net are in turn nets. In his work, Valk takes an object as a token in a unary elementary Petri net system, whereas the object itself is an elementary net system. Even if in the original Valk's proposal no dynamic changes are possible, and mobility is weakly supported, most extensions introduced afterward rely upon his idea.

[13] defined a class of high level Petri nets, called *reconfigurable nets*, which can dynamically modify their own structure by rewriting some of their components. Reconfigurable nets can be unfolded to a subclass of self-modifying Petri nets for which boundedness can be decided. Mobile and dynamic Petri nets [14] integrate Petri nets with RCHAM (Reflective Chemical Abstract Machine) based process algebra.

Tokens in self-modifying, mobile/dynamic and reconfigurable nets, are passive. To bridge the gap between tokens and active objects (agents) some variations on the theme of nets-within-nets have been proposed. In [15] objects are studied as high-level net tokens having an individual dynamical behavior. Object nets behave like tokens, i.e., they are lying in places and are moved by transitions. However, they may also change their state. Reference nets [16] are a flavor of high level Petri nets which provides dynamic creation of net instances, references to other nets/tokens, and communication via synchronous channels (net-inscriptions are in).

More recent proposals have some similarity with the work we are presenting. In [17], a dynamic architecture modeling is presented which allows active elements to be nested in arbitrary and dynamically changeable hierarchies, enabling the design of systems at different levels of abstractions, by using refinements of net models. In [18], the paradigm of *nets and rules as tokens* is introduced, where rules as tokens can be used, which permit the structure and behavior of P/T systems to be changed. The new concept is implemented using algebraic nets and graph transformations.

Most dynamic extensions of Petri nets set up new (hybrid) paradigms. While the expressive power has increased, the cognitive simplicity of Petri nets has decreased as well. As argued in [13], the intricacy of these proposals leaves little hope to obtain significant mathematical results and/or automated verification tools in a close future. The Reflective Petri nets approach is different, because it tries to achieve a satisfactory compromise between expressive power and analysis capability, through a rigorous application of reflection concepts in a consolidated high-level Petri Net framework.

7 Conclusions and Future Work

We have semi-formally introduced a state-transition semantics for reflective Petri nets, a formal layout based on classical Petri nets (Well formed Nets, and their unfolded counterpart) well suited to model adaptable/reconfigurable discrete-event systems. In particular, we have addressed a major topic related to recognizing equivalent system's evolutions, through the WN's symbolic state notion. We are planning to integrate the GreatSPN tool [19], that natively supports WN (and their stochastic extension, SWN), with new modules (plug-in's) for the graphical editing and the analysis/simulation of reflective Petri net models. We are defining a stochastic process for Reflective Petri nets, which is built from their state-transition graph, and which is in large part inspired to the SWN (GSPN) timed semantics.

References

[1] W. Reisig, *Petri Nets: An Introduction*, ser. EATCS Monographs on Theoretical Computer Science. Springer, 1985, vol. 4.

[2] L. Capra and W. Cazzola, "Self-Evolving Petri Nets," *Journal of Universal Computer Science*, vol. 13, no. 13, pp. 2002–2034, Dec. 2007.

[3] ——, "A Reflective PN-based Approach to Dynamic Workflow Change," in *Proceedings of the 9th International Symposium in Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'07)*. Timişoara, Romania: IEEE, on 26th-29th of Sep. 2007, pp. 533–540.

[4] K. Jensen and G. Rozenberg, Eds., *High-Level Petri Nets: Theory and Applications*. Springer-Verlag, 1991.

[5] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad, "On Well-Formed Coloured Nets and Their Symbolic Reachability Graph," in *Proceedings of the 11th International Conference on Application and Theory of Petri Nets,*, Paris, France, Jun. 1990, pp. 387–410.

[6] ——, "Stochastic Well-Formed Coloured Nets for Symmetric Modelling Applications," *IEEE Transactions on Computers*, vol. 42, no. 11, pp. 1343–1360, Nov. 1993.

[7] M. Ajmone Marsan, G. Conte, and G. Balbo, "A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems," *ACM Transaction on Computer Systems*, vol. 2, no. 2, pp. 93–122, May 1984.

[8] L. M. Kristensen, S. Christensen, and K. Jensen, "The practitioner's guide to coloured petri nets," *STTT*, vol. 2, no. 2, pp. 98–132, 1998.

[9] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad, "A Symbolic Reachability Graph for Coloured Petri Nets," *Theoretical Computer Science B (Logic, Semantics and Theory of Programming)*, vol. 176, no. 1& 2, pp. 39–65, Apr. 1997.

[10] C. Sibertin-Blanc, "The Hurried Philosophers," in *Concurrent Object-Oriented Programming and Petri Nets, Advances in Petri Nets*, ser. LNCS 2001, G. Agha, F. De Cindio, and G. Rozenberg, Eds. Springer, 2001, pp. 536–538.

[11] R. Valk, "Self-Modifying Nets, a Natural Extension of Petri Nets," in *Proceedings of the Fifth Colloquium on Automata, Languages and Programming (ICALP'78)*, ser. LNCS 62, G. Ausiello and C. Böhm, Eds. Udine, Italy: Springer, Jul. 1978, pp. 464–476.

[12] ——, "Petri Nets as Token Objects: An Introduction to Elementary Object Nets," in *Proceedings of the 19th International Conference on Applications and Theory of Petri Nets (ICATPN 1998)*, ser. LNCS 1420, J. Desel and M. Silva, Eds. Lisbon, Portugal: Springer, Jun. 1998, pp. 1–25.

[13] E. Badouel and J. Oliver, "Reconfigurable Nets, a Class of High Level Petri Nets Supporting Dynamic Changes within Workflow Systems," IRISA, IRISA Research Report PI-1163, Jan. 1998.

[14] A. Asperti and N. Busi, "Mobile Petri Nets," Università degli Studi di Bologna, Bologna, Italy, Technical Report UBLCS-96-10, May 1996.

[15] B. Farwer and D. Moldt, Eds., *Object Petri Nets, Process, and Object Calculi*. Hamburg, Germany: Universität Hamburg, Fachbereich Informatik, Aug. 2005.

[16] O. Kummer, "Simulating Synchronous Channels and Net Instances," in *Proceedings of the Workshop Algorithmen und Werkzeuge für Petrinetze*, ser. Forschungsberichte, J. Desel, P. Kemper, E. Kindler, and A. Oberweis, Eds., vol. 694. Universität Dortmund, Fachbereich Informatik, Oct. 1998, pp. 73–78.

[17] L. Cabac, M. Duvignau, D. Moldt, and H. Rölke, "Modeling Dynamic Architectures Using Nets-Within-Nets," in *Proceedings of the 26th International Conference on Applications and Theory of Petri Nets (ICATPN 2005)*, ser. LNCS 3536, G. Ciardo and P. Darondeau, Eds. Miami, FL, USA: Springer, Jun. 2005, pp. 148–167.

[18] K. Hoffmann, H. Ehrig, and T. Mossakowski, "High-Level Nets with Nets and Rules as Tokens," in *Proceedings of the 26th International Conference on Applications and Theory of Petri Nets (ICATPN 2005)*, ser. LNCS 3536, G. Ciardo and P. Darondeau, Eds. Miami, FL, USA: Springer, Jun. 2005, pp. 268–288.

[19] G. Chiola, G. Franceschinis, R. Gaeta, and M. Ribaudo, "GreatSPN 1.7: GRaphical Editor and Analyzer for Timed and Stochastic Petri Nets," *Performance Evaluation*, vol. 24, no. 1-2, pp. 47–68, Nov. 1995.