*Lorenzo Capra and Walter Cazzola. Evolving System's Modeling and Simulation through Reflective Petri Nets. In Proceedings of the 4th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE'09), pages 59–70, Milan, Italy, May 2009. INSTICC, Springer.*

# Evolving System's Modeling and Simulation through Reflective Petri Nets

Lorenzo Capra and Walter Cazzola

Department of Informatics and Communication,
Università degli Studi di Milano, Italy
`{capra|cazzola}@dico.unimi.it`

**Abstract.** The design of dynamic discrete-event systems calls for adequate modeling formalisms and tools 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 reflective Petri net 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 them 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. In this paper we provide the reflective Petri nets with a (labeled) state-transition graph semantics.

**keywords** Petri nets, evolving systems, state-transition semantics, symbolic techniques.

## 1 INTRODUCTION

Most existing discrete-event systems are subject to evolution during their lifecycle. As an example consider of mobile ad-hoc networks, adaptable software, business processes, and so on. Designing dynamic discrete-event systems calls for adequate modeling formalisms and tools. Unfortunately, well-established formalisms for discrete-event systems, such as classical Petri nets (Reisig, 1985), 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. This practice hampers system analysis, reuse and maintenance.

Reflective Petri nets (Capra and Cazzola, 2007*b*) were recently proposed as a formal model for dynamic discrete-event systems. 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 classical Petri nets should be preserved. Reflective nets were successfully applied to specify dynamic workflows (Capra and Cazzola, 2007*a*; Capra and Cazzola, 2008), in which the workflow's template can change according to business needs. On the line of (Qiu and Wong, 2007), the approach implements a smart transfer mechanisms of running workflow instances on the modified process template, which prevents already completed, bypassable tasks from being redone from scratch. It relies on checking the preservation of structural Petri net properties that permit an efficient characterization of workflow's soundness (van der Aalst and Basten, 2002) over the time.

On the perspective of implementing 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 (marking) definition that the particular flavor of Colored Petri nets (Jensen and Rozenberg, 1991) used for the meta-level is provided with, and represents the paper's main technical contribution.

The balance of the paper is as follows: background information are in sections 2,3, where the focus is put there on those elements directly connected to the paper's contribution, which is presented in section 4 where the state-transition semantics for Reflective Petri nets is defined. An application to an instance of dynamic system is presented in section 5. Finally section 6 is about work-in-progress. Assuming the readers have some basic knowledge about Petri nets, a semi-formal presentation is adopted, and a simple example is used. Unessential notions\notations are skipped, especially in introductory parts.

## 2 WN BASICS

The formalisms employed for the two layers (meta- and base-level) of the reflective layout are Well-formed Nets (WN) (Chiola, Dutheillet, Franceschinis and Haddad, 1997), a flavor of Colored Petri nets (CPN) (Jensen and Rozenberg, 1991), and their unfolding, an extension of ordinary Place/Transition nets (Reisig, 1985) respectively. This choice has revealed convenient for two main reasons: i) the behavior of reflective Petri nets can be formally stated in terms of classical Petri nets; ii) the symbolic state representation the WN formalism is provided with 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.ding, an extension of ordinary Place/Transition nets (Reisig, 1985) respectively. This choice has revealed convenient for two main reasons: i) the behavior of reflective Petri nets can be formally stated in terms of classical Petri nets; ii) the symbolic state representation the WN formalism is provided with 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 WNs (SWN) (Chiola, Dutheillet, Franceschinis and Haddad, 1993), and their unfolding, that is Generalized Stochastic Petri nets (GSPN) (Ajmone Marsan, Conte and Balbo, 1984), 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 Continuous Time Markov Chain (CTMC).*

## 2.1 WN color annotations

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* $\mathbf{M}$ 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* $C_{\mathrm{i}}$, that may be in turn partitioned into *static subclasses* $C_{\mathrm{i,k}}$. 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 each other by means of permutations on color classes that preserve static subclasses. A SM is formally expressed in terms of dynamic subclasses.

## 2.2 The Symbolic Marking (SM) notion.

The definition of a SM (denoted $\widehat{\mathbf{M}}$) (Chiola et al., 1997) 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 $D_{\mathrm{i}}$ and $s_{\mathrm{i}}$ denote the set of dynamic subclass of $C_{\mathrm{i}}$ (in $\widehat{\mathbf{M}}$), and the number of static subclasses of $C_{\mathrm{i}}$ (if $C_{\mathrm{i}}$ is not split then $s_{\mathrm{i}} = 1$). The j-th dynamic subclass of $C_{\mathrm{i}}$, $Z^{\mathrm{i}}_{\mathrm{j}} \in D_{\mathrm{i}}$, refers to a static subclass, denoted $d(Z^{\mathrm{i}}_{\mathrm{j}})$, $1 \leq d(Z^{\mathrm{i}}_{\mathrm{j}}) \leq s_{\mathrm{i}}$, and has an associated cardinality $|Z^{\mathrm{i}}_{\mathrm{j}}|$, i.e., it represents a parametric set of colors (in the sequel we shall consider cardinality one dynamic subclasses). It must hold, for each $k : 1 \ldots s_{\mathrm{i}}$, $\sum_{\mathrm{j}:d(Z^{\mathrm{i}}_{\mathrm{j}})=k} \left|Z^{\mathrm{i}}_{\mathrm{j}}\right| = \mid C_{\mathrm{i,k}} \mid$.

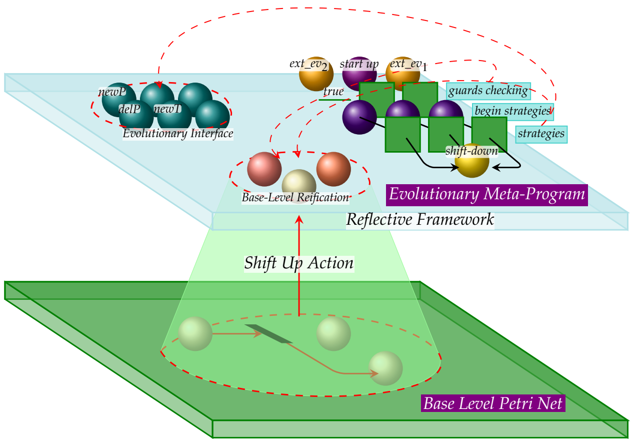

**Fig. 1.** Snapshot of the reflective PN model.

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 (Chiola et al., 1997) 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 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 (Fig. 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 to the base-level at the end of a meta-computation (*shift-down action*)

The meta-program is implicitly activated (*shift-up action*) and 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. 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 change caused in the meanwhile to the base-level reification is 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.

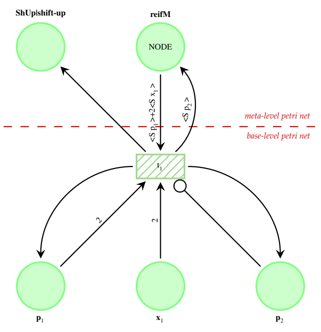

**Fig. 2.** Reification implemented at Petri nets level.

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*b*). Let us only outline some essential points:

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 strategies represent the variable component.
2. 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 rises to the meta-model, called hereafter *meta-level PN*.
3. The reification-set is formed by the following colored places: `reifN`, `reifA`, `reifΠ`, `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 the most important, there is a one-to-one correspondence, formalized by a bijection, between reifications and P/T nets.

4. The initial reification, from time to time, refers to the base-level Petri net modeling the initial system configuration.
5. The shift-up action and the reification update after any base-level change of state, are implemented in transparent way at net level, by suitably connecting any base-level PN transition to place `reifM` , that 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 changes of state at the base-level Petri net are instantaneously reproduced on the reification, conceptually maintaining base-level 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 to the base-level (e.g., $x_1$), by means of WN constant functions.
6. The shift-down action, 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 replaces the (current) base-level PN 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 the evolution.

## 4 STATE-TRANSITION SEMANTICS FOR REFLECTIVE PN

The adoption of WN formalism (Chiola et al., 1997) for the meta-program permits an effective definition of behavior, in terms of state-transitions. In particular, the WN symbolic marking notion is exploited to identify equivalent states a Reflective Petri net model (possibly) reaches during the evolution.

On the light of the causal connection established between base- and meta-level, the behavior of a Reflective Petri net model between any meta-level activations is fully described in terms of a WN model, the meta-level PN, which includes (better, is suitably connected to) an uncolored part, the base-level PN. That 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 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

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

There is nothing to do but consider the case where $m_f$ is a marking enabling the pseudo-transition `shiftdown`: then,

$$\mathbf{M}_f \xrightarrow{\texttt{shiftdown}} \mathbf{M}'_0,$$

$\mathbf{M}'_0$ 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 `shiftdown` 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 tricky. 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:

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

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 each other. 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.*

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

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

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

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

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

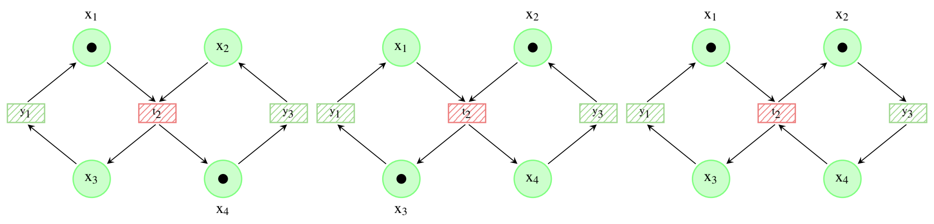

**Fig. 3.** Equivalent base-level Petri Nets

Of course $\widehat{\mathbf{M}}'_i \equiv \widehat{\mathbf{M}}_i$, as well as any intermediate marking crossed by $\sigma$.

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. 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 and a new marking are set.

Nevertheless, the three base-level configurations are equivalent, according to definition 2. It is sufficient to take a look at their respective reifications, 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:

$$\begin{aligned}
\widehat{\mathbf{M}}(\texttt{reifN}) &= y_1+y_3+t_2+x_1+x_2+x_3+x_4,\\
\widehat{\mathbf{M}}(\texttt{reifM}) &= x_1+x_4,\\
\widehat{\mathbf{M}}(\texttt{reifA}) &= \langle x_1,t_2\rangle+\langle t_2,x_3\rangle+\langle x_3,y_1\rangle+\langle y_1,x_1\rangle+\langle x_2,t_2\rangle+\langle t_2,x_4\rangle+\langle x_4,y_3\rangle+\langle y_3,x_2\rangle
\end{aligned}$$

and

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

respectively.

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

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

thus, they are equivalent. With similar arguments we might 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:

$$\begin{aligned}
\widehat{\mathbf{M}}''(reifN) &= y_1+y_3+t_2+x_1+x_2+x_3+x_4\\
\widehat{\mathbf{M}}''(reifM) &= x_1+x_2\\
\widehat{\mathbf{M}}''(reifA) &= \langle x_1,t_2\rangle+\langle t_2,x_3\rangle+\langle x_3,y_1\rangle+\langle y_1,x_1\rangle+\langle x_2,y_3\rangle+\langle y_3,x_4\rangle+\langle x_4,t_2\rangle+\langle t_2,x_2\rangle
\end{aligned}$$

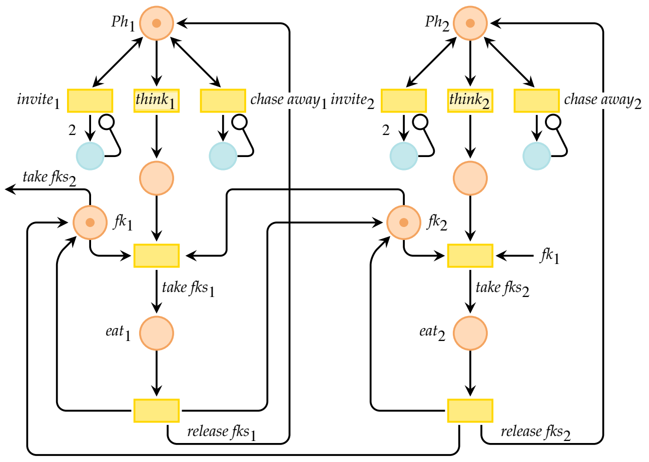

**Fig. 4.** Base-level Petri net modeling the initial hurried philosopher's table.

$\widehat{\mathbf{M}}$ and $\widehat{\mathbf{M}}''$ can be obtained from each other by the following permutation:

$$\{x_2 \leftrightarrow x_4\}$$

The canonical representative of these equivalent states is $\widehat{\mathbf{M}}$.

## 5 APPLYING THE SEMANTICS

The (symbolic) state-transition semantics of Reflective Petri nets has been tested on the *hurried philosophers* problem (Sibertin-Blanc, 2001), a variant of the well known dining philosophers problem that introduces a high dynamism and mobility degree: philosophers can join and leave the table upon invitation. The version here considered meets the following requirements:

- (at least) two philosophers are initially sitting at the table;
- each philosopher can eat only when he contemporary gets both adjacent forks, according to the classical problem's formulation;
- those philosophers sitting at the table that are not eating have the following additional capabilities:
  - they can invite another philosopher (arbitrarily chosen) to join the table and sit down at his side, if the table capacity has not been exceeded yet;
  - they can ask one of the adjacent philosophers to leave the table (if more than two are currently sitting);
- each philosopher is going around with his own fork; when he joins the table he keeps the fork, when he leaves he brings the fork with him.

Figure 4 shows the base-level Petri net modeling the initial configuration, where two philosophers are sitting on the dining table. The invitation and chasing away capabilities provoke the evolution of this simple model. Any arrivals/departures of philosophers cause a local reconfiguration of the system's topology, that consists of carefully changing adjacencies and fork sharing in a consistent way. All that is completely managed at meta-program level, as specified in (Capra and Cazzola, 2007*b*). Table 1 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 action. 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.

## 6 CONCLUSION

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 unfolding) well suited to model dynamic 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 (Chiola, Franceschinis, Gaeta and Ribaudo, 1995), 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 studying the possibility of defining a timed semantics for Reflective Petri nets, which should be in large part inspired by SWN (GSPN) timed semantics. Finally, we are considering possible applications of reflective Petri nets for the simulation of protocols for mobile, ad-hoc networks.

## References

Ajmone Marsan, M., Conte, G. and Balbo, G. (1984), 'A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems', *ACM Transaction on Computer Systems* **2**(2), 93–122.

Capra, L. and Cazzola, W. (2007*a*), 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)', IEEE, Timişoara, Romania, pp. 533–540.

Capra, L. and Cazzola, W. (2007*b*), 'Self-Evolving Petri Nets', *Journal of Universal Computer Science* **13**(13), 2002–2034.

**Table 1.** Symbolic vs. ordinary state-space size; — data not available

| # philosophers | symbolic states | ordinary states |
|---:|---:|---:|
| 3 | 423 | 2567 |
| 4 | 2843 | 374809 |
| 5 | 23560 | 1765836 |
| 6 | 147812 | 96905034 |
| 7 | 960345 | — |
| 8 | 4767385 | — |
| 9 | 12097086 | — |

Capra, L. and Cazzola, W. (2008), Evolutionary Design through Reflective Petri Nets: an Application to Workflow, *in* 'Proceedings of the IASTED International Conference on Software Engineering (SE'08)', ACTA Press, Innsbruck, Austria.

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), 1343–1360.

Chiola, G., Dutheillet, C., Franceschinis, G. and Haddad, S. (1997), 'A Symbolic Reachability Graph for Coloured Petri Nets', *Theoretical Computer Science B (Logic, Semantics and Theory of Programming)* **176**(1& 2), 39–65.

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* **24**(1-2), 47–68.

Jensen, K. and Rozenberg, G., eds (1991), *High-Level Petri Nets: Theory and Applications*, Springer-Verlag.

Qiu, Z.-M. and Wong, Y. S. (2007), 'Dynamic Workflow Change in PDM Systems', *Computers in Industry* **58**(5), 453–463.

Reisig, W. (1985), *Petri Nets: An Introduction*, Vol. 4 of *EATCS Monographs on Theoretical Computer Science*, Springer.

Sibertin-Blanc, C. (2001), The Hurried Philosophers, *in* G. Agha, F. De Cindio and G. Rozenberg, eds, 'Concurrent Object-Oriented Programming and Petri Nets, Advances in Petri Nets', LNCS 2001, Springer, pp. 536–538.

van der Aalst, W. M. P. and Basten, T. (2002), 'Inheritance of Workflows: An Approach to Tackling Problems Related to Change', *Theoretical Computer Science* **270**(1-2), 125–203.