Chapter 19
A Quotient-Graph for the Analysis of Reflective 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 · quotient-graph

19.1 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 [13], 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.

L. Capra (✉)
Department of Informatics and Communication (D.I.Co),
Università degli Studi di Milano, MI Italy 20139
e-mail: capra@dico.unimi.it

S.-I. Ao and L. Gelman (eds.), *Electronic Engineering and Computing Technology*,
Lecture Notes in Electrical Engineering 60, DOI 10.1007/978-90-481-8776-8_19,

Reflective Petri nets [5] have been recently proposed as design framework for dynamic discrete-event systems, and successfully applied to dynamic workflows [4]. 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. With respect to other dynamic extensions of Petri nets appeared in last decade, which set up new (hybrid) paradigms [2, 3, 9, 12], the Reflective Petri nets approach tries to achieve a satisfactory compromise between expressive power and analysis capability, through a rigorous application of reflection concepts in a consolidated Petri net framework.

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 [11] 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 19.2 and 19.3. 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 19.4). An application of the semantics to a dynamic system taken from literature is summarized in Section 19.5. Related works are mentioned and discussed in Section 19.6. Finally Section 19.7 is about work-in-progress.

19.2 WN's Basic Notions

The formalisms employed for the two levels (meta- and base-) of the reflective layout are Well-formed Nets (WN) [6], a flavor of Colored Petri nets (CPN) [11], and their unfolded counterpart, an extension of ordinary Place/Transition nets [13], 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 state-transition; secondly, the symbolic state notion peculiar of WN makes it possible to efficiently recognize equivalent base-level's evolutions.

While retaining CPN's expressive power, WNs are characterized by a structured syntax, exploited by efficient analysis algorithms.This section does not present all the features of WNs, for which the reader can refer to [6], just introduces them informally, focusing on the symbolic marking definition. Unlike CPNs, WNs include priority levels for transition and inhibitor arcs. These features enhance the formalism expressiveness and are helpful to represent the transactional execution of evolutionary strategies.

As in CPN, 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 there exist different *color instances* of a given transition. A *marking* $\mathbf{M}$ maps every place p to a multiset on the respective color domain $\mathcal{C}(p)$. The projection of $\mathbf{M}$ to a subset P' is denoted $\mathbf{M}[P']$. Any arc connecting p to a transition t is labeled by a function mapping every element of $\mathcal{C}(t)$ to a multiset on $\mathcal{C}(p)$.

SWN color domains are Cartesian products of *basic color classes* C_{i}. A class C_{i} may be in turn partitioned into *static subclasses* $C_{\mathrm{i,k}}$. The idea is that objects in a subclass are indistinguishable from one another.

19.2.1 The Symbolic Marking

The Symbolic Marking [7] provides syntactical equivalence relation on ordinary WN markings: two markings belong to the same SM if and only if they can be obtained from one another by means of a permutation of colors that preserve static subclasses. A SM (denoted $\widehat{\mathbf{M}}$) is formally expressed in terms of *dynamic subclasses*, and specifies the distribution of symbolic colors (tuples built of dynamic subclasses) over the WN places.

Dynamic subclasses define a parametric partition of color classes preserving the static partition: let Z_{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} ($s_{\mathrm{i}} \geq 1$). The j-th dynamic subclass of C_{i}, $z^{\mathrm{i}}_{\mathrm{j}} \in Z_{\mathrm{i}}$, 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 a cardinality $|z^{\mathrm{i}}_{\mathrm{j}}|$, i.e., it represents a parametric set of colors. It must hold, $\forall k : 1...s_{\mathrm{i}}$ $\sum_{\mathrm{j}:d(z^{\mathrm{i}}_{\mathrm{j}})=k} \left|z^{\mathrm{i}}_{\mathrm{j}}\right| = \mid C_{\mathrm{i,k}} \mid$.

The SM canonical form [7], based on a lexicographic ordering and a minimization of dynamic subclass distribution over the places, provides a way to uniquely represent an SM.

19.3 Reflective Petri Nets Layout

The *Reflective Petri nets* [5] approach relies on a logical layout divided in two levels. 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 level, 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 upon occurrence of certain conditions/events.

The meta-level acts on a representative of the base-level, called *reification*, which is formalized by a colored marking. The reification is used by the meta-program to

observe (*introspection*) and manipulate (*intercession*) the base-level PN. Changes to the reification are reflected down to the base-level at the end of a meta-computation (*shift-down*).

The meta-level is implicitly activated (*shift-up*) at any base-level change of state. Then a strategy is selected depending on whether (a) the base-level has entered a given condition, (b) and/or any external events (simulated at meta-level) have occurred. The ability of specifying arbitrary selection conditions enhances the flexibility of the reflective layout.

The *reflective framework* is another high-level Petri net component, somehow similar to a transparent meta-layer, which is in charge of implementing base-level's introspection and intercession. The framework has a fixed layout, formed by higher-priority transitions. Intercession is performed in terms of a minimal, complete set of low-level operations (the *evolutionary interface*): addition/removal of nodes and arcs, change of transition priorities (structural changes), free moving of tokens over-all the base-level PN places (state changes). If one such operation fails, the meta-program as a whole is restarted and any changes caused in the meanwhile to the reification are discarded. Trying to delete a yet not existing node is an example of failure In other words, the evolutionary strategies have a transactional semantics. After a strategy's succeeding run, changes are reflected down to the base-level Petri net.

A designer is provided with a tiny ad-hoc language, originally inspired to Hoare's CSP, to specify his/her own strategy in a simple way, without any skills in high-level Petri net 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, varying from a non deterministic choice, to a static assignment of priorities. According to the reflective paradigm, the base-level is unaware of the meta-program. The system designer may freely decide, using priority, to block the base-level while the meta-program is active, or to leave it running. It may also define local influence areas for some strategies, by (temporarily) locking corresponding portions of the base-level.

Let us only outline here some essential points about the interaction between base- and meta-levels:

1. The reflective framework and the meta-program share two sets of boundary colored places, denoted *reification* set and *evolutionary interface* in the sequel. Their composition, through a simple superposition of shared places, gives rise to the meta-model, called *meta-level PN*.
2. The reification is a *well-defined* marking of the reification set `Reif` formed by $\{\texttt{reifP}, \texttt{reifT}, \texttt{reifA}, \texttt{reif}\Pi, \texttt{reifM}\}$. Such places encode structure (nodes, connections, and priorities) and current state of the base-level PN, respectively. Their color domains are built of basic classes *Place*, *Tran*, which are (logically) unbounded repositories holding all *potential* base-level nodes (they must contain the nodes of the initial base-level Petri net). We have: $\mathcal{C}(\texttt{reifP})$, $\mathcal{C}(\texttt{reifM})$: *Place*; $\mathcal{C}(\texttt{reifT})$, $\mathcal{C}(\texttt{reif}\Pi)$: *Tran*; $\mathcal{C}(\texttt{reifA})$: $\mathit{Place} \times \mathit{Tran} \times \{i, o, h\}$.

3. The isomorphism between base-level nets and reification is formalized by a bijection φ. For example a net formed by places $\{p_1, p_2, \ldots\}$, transitions $\{t_1, t_2, \ldots\}$ having priority levels $0, 1, \ldots$ respectively, by an input arc (p_1, t_1) of weight 2, an output arc (t_1, p_2) of weight 1 .., whose current marking is $\mathbf{m}(p_1) = 2$, $\mathbf{m}(p_2) = 1$, is reified as: $\mathbf{M}(\texttt{reifP}) = p_1 + p_2 + \ldots$, $\mathbf{M}(\texttt{reifT}) = t_1 + t_2 + \ldots$, $\mathbf{M}(\texttt{reif}\Pi) = t_2 + \ldots$, $\mathbf{M}(\texttt{reifA}) = 2 \cdot \langle p_1, t_1, i \rangle + \langle p_1, t_2, o \rangle + \ldots$, $\mathbf{M}(\texttt{reifM}) = 2 \cdot p_1 + p_2$.
4. A back-up copy $\texttt{Reif}_{back}$ of set Reif is kept. Evolutionary strategies work on Reif: if an operation fails, then the contents of $\texttt{Reif}_{back}$ are copied back to Reif, and the control passes to the base-level.
5. The shift-up is implemented in transparent way at net-level, by connecting every base-level transition to the place(s) reifM ($\texttt{reifM}_{back}$) by means of colored arcs. The resulting model is denoted *base-meta PN*. Consider transition t_1 of the above example: its firing makes two colors p_1 and one color p_2 be withdrawn/added from/to reifM ($\texttt{reifM}_{back}$), respectively. Base-level changes of state are thus instantaneously mirrored on the reification, maintaining base-level's unawareness of the meta-level.
6. The shift-down is emulated by a homonym highest-priority meta-transition of the meta-level PN.

19.4 State-Transition Semantics for Reflective Nets

The causal connection between base- and meta-level makes it possible to formalize the behavior of Reflective Petri nets in terms of WN state-transitions:

Definition 19.1 (Reflective Petri net state). A state of a Reflective Petri net is a marking $\mathbf{M}_\mathrm{i}$ of the base-meta PN.

Let PN_0 be the (marked) base-level Petri net which models the initial system. Assume it has been connected to the meta-level. The initial state of the corresponding Reflective Petri net is obtained setting: $\mathbf{M_0}[\texttt{Reif}] = \mathbf{M_0}[\texttt{Reif}_{back}] = \varphi(PN_0)$.

Let t_c be any transition (color instance) of the base-meta PN, other than shiftdown. If t_c is enabled in $\mathbf{M}_\mathrm{i}$, according to the ordinary enabling rule, and $\mathbf{M}_\mathrm{j}$ is the marking reached upon the firing, we have the state-transition:

$$\mathbf{M}_\mathrm{i} \xrightarrow{t_\mathrm{c}} \mathbf{M}_\mathrm{j}$$

Only one case must be treated apart. Let *shift-down* be enabled in $\mathbf{M}_\mathrm{i}$, according to the ordinary firing rule. Then:

$$\mathbf{M}_\mathrm{i} \xrightarrow{shift-down} \mathbf{M}'_\mathbf{0},$$

$\mathbf{M}'_0$ being the marking of the base-meta PN obtained by firing *shift-down* in the ordinary way, making the contents of $\texttt{Reif}_{back}$ be updated to `Reif`, finally (side-effect), replacing the current base-level PN with $PN' = \varphi^{-1}(\mathbf{M}_\mathrm{i}[\texttt{Reif}])$ and connecting PN' to the meta-level.

19.4.1 Handling Equivalent Evolutions

The just introduced state-transition semantics defines precisely the untimed behavior of a reflective Petri net, but suffers from two evident drawbacks affecting efficiency and effectiveness. First, the notion of state is exceedingly redundant, comprising a part, the meta-level, which outs the functional specification of a system. Secondly, there is no way of recognizing whether the system dynamics/evolution leads to equivalent conditions. The latter question is critical: the ability of deciding about finiteness and strongly-connectedness (strictly related to the ability of recognizing equivalences) is in fact mandatory for any techniques based on state-space inspection.

Recognizing equivalences in an evolving system is tricky. It may happen that after a series of transformations the base-level comes back to the original condition (state). Even more likely, the internal dynamics of the evolving system might lead to equivalent conditions. The problem is tackled by resorting to the symbolic marking notion, peculiar of WN, and the base-level reification at the meta-level.

The modeler, on his/her needs may define a *logical* partition of classes *Place*, *Tran*, possibly different from the completely split partition (implicitly) adopted when setting up the base-meta PN:

$$Place = P_1 \cup P_2 \cup \ldots P_\mathrm{k} \quad Tran = T_1 \cup T_2 \cup \ldots T_\mathrm{n}$$

The idea is simple: elements belonging to a subclasses P_i (T_j) denote indistinguishable base-level nodes, which might be freely permuted, without altering the model's semantics. Those nodes that, for any reasons, must preserve their identity during evolution, will correspond to cardinality one subclasses. The default logical partition is that in which all places/transitions can be permuted. Of course the evolutionary strategies refer to the logical partition of base-level nodes.

The causal connection between base- and meta- levels establishes an exact correspondence (at any instant) between the current base-level PN and the contents of $\texttt{Reif}_{back}$. On the light of that, we state the following state-equivalence notion, in which we refer to the logical partition of *Place* and *Tran*.

Definition 19.2 (state equivalence). Let $\widehat{\mathbf{M}}_\mathrm{i}$ be the symbolic marking obtained from $\mathbf{M}_\mathrm{i}[\texttt{Reif}_{back}]$ replacing every $p_\mathrm{i} \in P_\mathrm{k}$ ($t_\mathrm{j} \in T_\mathrm{l}$) with a corresponding dynamic subclass z^1_i (z^2_j), $d(z^1_\mathrm{i}) = k$ ($d(z^2_\mathrm{j}) = l$), $|z^1_\mathrm{i}|$ ($|z^2_\mathrm{j}|$) $= 1$. Then $\mathbf{M}_\mathrm{i} \equiv \mathbf{M}_\mathrm{j}$ if and only if $\widehat{\mathbf{M}}_\mathrm{i} \equiv \widehat{\mathbf{M}}_\mathrm{j}$.

$\widehat{\mathbf{M}}_i$ represents an equivalence class of states (Def.19.1), so we shall use the notation $\mathbf{M} \in \widehat{\mathbf{M}}_i$. The state-transition notion is redefined accordingly.

Definition 19.3 (visible state-transition). Let σ be a finite sequence of meta-level transition color instances other than `shiftdown` (σ possibly empty). Then $\mathbf{M}_i \xrightarrow{t} \mathbf{M}_j$, if and only if t is either `shiftdown` or a base-level transition, and there exist $\sigma, \mathbf{M}'_i$ s.t. $\mathbf{M}_i \xrightarrow{\sigma} \mathbf{M}'_i \xrightarrow{t} \mathbf{M}_j$ (according to the above definition).

$\mathbf{M}'_i$, as well as any intermediate marking crossed by σ, are equivalent to $\mathbf{M}_i$. Visible state-transitions are caused by the occurrence of either *shiftdown*, or any base-level transition. Meta-level transition sequences (σ) are not visible to an external observer.

We call reachable a state $\mathbf{M}_i$ such that $\mathbf{M_0} \xrightarrow{t_1} \mathbf{M_1} \xrightarrow{t_2} \ldots \mathbf{M}_i$. We say $\widehat{\mathbf{M}}_i$ reachable if and only if any $\mathbf{M} \in \widehat{\mathbf{M}}_i$ is.

Lemma 19.4. *Let* $t \in T_k$, $\mathbf{M}_i \xrightarrow{t} \mathbf{M}_j$. *Then:*

$\forall \mathbf{M} \in \widehat{\mathbf{M}}_i \; \exists t' \in T_k, \; \mathbf{M}' \in \widehat{\mathbf{M}}_j \; \mathbf{M} \xrightarrow{t'} \mathbf{M}'$

$\forall \mathbf{M}' \in \widehat{\mathbf{M}}_j \; \exists t' \in T_k, \; \mathbf{M} \in \widehat{\mathbf{M}}_i \; \mathbf{M} \xrightarrow{t'} \mathbf{M}'$

Thanks to the above lemma we can build a *quotient-graph* in which nodes are the reachable $\{\widehat{\mathbf{M}}_i\}$, and there is a labeled arc $\widehat{\mathbf{M}}_i \xrightarrow{T_k} \widehat{\mathbf{M}}_j$ if and only if there exist $t \in T_k, \mathbf{M} \in \widehat{\mathbf{M}}_i, \mathbf{M}' \in \widehat{\mathbf{M}}_j$, s.t. $\mathbf{M} \xrightarrow{t} \mathbf{M}'$.

If the meta-level PN never enters a deadlock or a livelock, then the *liveness* and *reachability* properties of the original state-transition graph are preserved.

19.5 The Dynamic Philosophers Example

The (symbolic) state-transition semantics of Reflective Petri nets has been tested on a variant of the well known dining philosophers problem which introduces a high dynamism [14]. The version here considered meets the following requirements:

- Two philosophers initially sit on the table.
- A philosopher can eat only when he/she simultaneously picks up the pair of adjacent forks, one of which is owned by the philosopher.
- A philosopher sitting on the table has two additional faculties, both requiring that the owned fork is currently available.
  - He/she can invite a colleague which is outside to join the table, sharing with him/her the owned fork.
  - He/she can leave the table, if there are at least three philosophers sited on it.
- Each philosopher is going around with his/her own fork.

The base-level Petri net representing the starting condition is depicted in Fig. 19.1. We observe that the functional aspects are described in detail, while

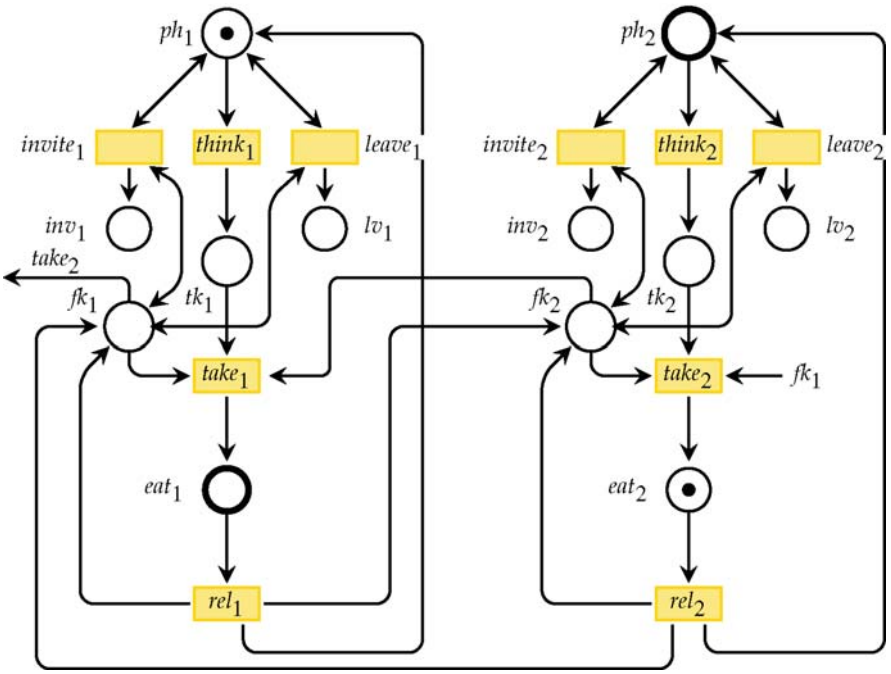

Fig. 19.1 Dynamic philosopher's base-level Petri net

the dynamic features are only sketched (transitions $invite_i$, $leave_i$), thus keeping the model as simple as possible. Any invitation/leaving intents activate the meta-program, which consequently implements two different strategies.

The logical partition of base-level nodes groups places/transition playing a similar role:

$$Place = Ph \cup Fork \cup Lv \cup Inv \cup \ldots \quad Tran = Invite \cup Think \cup Leave \cup \ldots$$

where $Ph = \{ph_i\}$, $Fork = \{fk_i\}$, $Lv = \{lv_i\}$,..$Think = \{th_i\}$, etc.

According to Def. 19.2 the depicted base-level net, and that having the same structure, but places eat_1, ph_2 marked instead of eat_2, ph_1, are equivalent (reachable) states of the corresponding Reflective Petri net. They can be obtained from one another by the permutation:

$$\{ph_1 \leftrightarrow ph_2, eat_1 \leftrightarrow eat_2\}$$

The *leaving* strategy is informally described in Table 19.1. The strategy is divided into an *introspection* step, which consists of checking a logical condition on the base-level reification, followed, if the check is positive, by an *intercession* phase.

Table 19.1 Leaving strategy description

introspection
There exist a marked place $l1 : Lv$ and at least 3 places of type Ph

intercession
The philosopher $ph1 : Ph$ who issued the request is identified:
Let $f1$ be the owned fork, $f2$ be the other fork used by $ph1$ ($f1, f2 : Fork$);
The philosopher $ph3$ sharing fork $f1$ with $ph1$ is identified as well:
Let $tk3 : Take$, $rel3 : Rel$ be the transitions modeling the pick-up and the release of forks by $ph3$, respectively;
$ph3$ is connected to $f2$ through a new input arc ($f2, tk3$), and a new output arc ($rel3, f2$);
The philosopher $ph1$ (meant as a the whole subnet) is removed, together with $f1$;
Place $l1$ is emptied;

Table 19.2 Symbolic vs. ordinary state-space size

#Philosophers	Symbolic states	Ordinary states
3	42	256
4	284	37,489
5	2, 356	176,583
6	14, 712	5,905,034
7	96, 035	*not av.*
8	476, 785	*not av.*
9	1, 207, 086	*not av.*
10	2, 978, 896	*not av.*

Symbols used in the description correspond to typed variables of the strategy specification language, which are bound from time to time to color instances.

An evidence of the effectiveness of the quotient graph based on the equivalent states notion (Def. 19.2), which comes to be live, versus the ordinary state-transition graph, is given in Table 19.2. Only visible changes of state involving the base-level are numbered. The experiment was conducted using the GreatSPN tool, with a script emulating the shift-down effect. The first column reports the problem size, i.e., the table capacity. We can appreciate a sensible reduction of the number of reached states also for small sizes, due to the high symmetry exhibited by the system during evolution. Some data about time/memory saving, not reported for the lack of space, confirm the effectiveness of the approach.

19.6 Related Works

Many efforts have been devoted in trying to extend Petri nets with dynamical features. In [15], 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 [16], 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.

Badouel and Oliver [2] 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 [1] 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 [9] 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 [12] 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 `Java`).

More recent proposals have some similarity with the work we are presenting. In [3], 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 [10], the paradigm of *nets and rules as tokens* is introduced, 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 [2], 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.

19.7 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 [8], that natively supports WN and their stochastic extension, SWN, with new modules for the graphical editing and the

analysis/simulation of reflective Petri net models. For that purpose we are defining a stochastic process for Reflective Petri nets, in large part inspired to the SWN (GSPN) timed semantics.

References

1. Asperti, A., Busi, N.: Mobile Petri Nets. Technical Report UBLCS-96-10, Università degli Studi di Bologna, Bologna, Italy (1996)
2. Badouel, E., Oliver, J.: Reconfigurable Nets, a Class of High Level Petri Nets Supporting Dynamic Changes within Workflow Systems. IRISA Research Report PI-1163 IRISA (1998)
3. Cabac, L., Duvignau, M., Moldt, D., Rölke, H.: Modeling dynamic architectures using nets within nets. In: Ciardo, G., Darondeau, P. (eds.) Proceedings of the 26th International Conference on Applications and Theory of Petri Nets (ICATPN 2005), LNCS 3536, pp. 148–167. Miami, FL, Springer (2005)
4. Capra, L., Cazzola, W.: 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), pp. 533–540. Timişoara, Romania, IEEE (2007a)
5. Capra, L., Cazzola, W.: Self-evolving Petri nets. J. Univ. Comp. Scie. **13**(13), 2002–2034 (2007b)
6. Chiola, G., Dutheillet, C., Franceschinis, G., Haddad, S.: On well-formed coloured nets and their symbolic reachability graph. In: Proceedings of the 11th International Conference on Application and Theory of Petri Nets, pp. 387–410. Paris, France (1990)
7. Chiola, G., Dutheillet, C., Franceschinis, G., Haddad, S.: A symbolic reachability graph for coloured Petri nets. Theor. Comput. Sci. B (Logic, Semantics and Theory of Programming) **176**(1& 2), 39–65 (1997)
8. Chiola, G., Franceschinis, G., Gaeta, R., Ribaudo, M.: GreatSPN 1.7: graphical editor and analyzer for timed and stochastic Petri nets. Perform. Evaluation **24**(1–2), 47–68 (1995)
9. Farwer, B. and Moldt, D. (eds.) Object Petri nets, process, and object calculi. Hamburg, Germany, Universität Hamburg, Fachbereich Informatik (2005)
10. Hoffmann, K., Ehrig, H., Mossakowski, T.: High-Level Nets with Nets and Rules as Tokens. In: Ciardo, G., Darondeau, P. (eds.) Proceedings of the 26th International Conference on Applications and Theory of Petri Nets (ICATPN 2005), LNCS 3536, pp. 268–288. Springer, Miami, FL (2005)
11. Jensen, K., Rozenberg, G. (eds.): High-Level Petri Nets: Theory and Applications. Springer, Berlin (1991)
12. Kummer, O.: Simulating synchronous channels and net instances. In: Desel, J., Kemper, P., Kindler, E., Oberweis, A. (eds.) Proceedings of the Workshop Algorithmen und Werkzeuge für Petrinetze, vol. 694 of *Forschungsberichte*, pp. 73–78. Universität Dortmund, Fachbereich Informatik (1998)
13. Reisig, W.: Petri Nets: An Introduction, vol. 4 of EATCS Monographs on Theoretical Computer Science. Springer, Berlin (1985)
14. Sibertin Blanc, C.: The hurried philosophers. In: Agha, G., De Cindio, F., Rozenberg, G. (eds.) Concurrent Object-Oriented Programming and Petri Nets, Advances in Petri Nets, LNCS 2001, pp. 536–538. Springer, Berlin (2001)
15. Valk, R.: Self-modifying nets, a natural extension of Petri nets. In: Ausiello, G., Böhm, C. (eds.) Proceedings of the Fifth Colloquium on Automata, Languages and Programming (ICALP'78), LNCS 62, pp. 464–476. Springer, Udine, Italy (1978)
16. Valk, R.: Petri nets as token objects: an introduction to elementary object nets. In: Desel, J., Silva, M. (eds.) Proceedings of the 19th International Conference on Applications and Theory of Petri Nets (ICATPN 1998), LNCS 1420, pp. 1–25. Springer, Lisbon, Portugal (1998)