Structural and Dynamic Changes in Concurrent Systems: Reconfigurable Petri Nets

Marisa Llorens and Javier Oliver

Abstract—The aim of this work is the modeling and verification of concurrent systems subject to dynamic changes using extensions of Petri nets. We begin by introducing the notion of net rewriting system. In a net rewriting system, a system configuration is described as a Petri net and a change in configuration is described as a graph rewriting rule. We show that net rewriting systems are Turing powerful, that is, the basic decidable properties of Petri nets are lost and, thus, automatic verification in not possible for this class. A subclass of net rewriting systems are reconfigurable Petri nets. In a reconfigurable Petri net, a change in configuration amounts to the modification of the flow relations of the places in the domain of the involved rule according to this rule, independently of the context in which this rewriting applies. We show that reconfigurable Petri nets are formally equivalent to Petri nets. This equivalence ensures that all the fundamental properties of Petri nets are still decidable for reconfigurable Petri nets and this model is thus amenable to automatic verification tools. Therefore, the expressiveness of both models is the same, but, with reconfigurable Petri nets, we can easily and directly model systems that change their structure dynamically.

Index Terms—Theory of computation, computation by abstract devices, models of computation, relations between models, modes of computation, parallelism and concurrency.

1 Introduction

A Petri net [19], [20] is a formalism to model, analyze, simulate, control, and evaluate the behavior of distributed and concurrent systems. This formalism, however, does not offer a direct way to address some modeling issues such as dynamic changes, multiple operating modes of operations, etc. Extensions of Petri nets have been designed to allow for an easy formalization of such features. The benefit in terms of modeling power is usually at the expense of a loss in decidable properties. A trade off needs to be found between expressiveness and computability. The fundamental goal of this work is the modeling and verification of concurrent systems that are subject to dynamic changes. It is important that the mechanism for handling dynamic changes in such systems be explicitly represented inside the model so that, at each stage of product development, designers can experiment with the effects of structural changes (e.g., by using prototypes). This means that structural changes are taken into account from the very beginning of the design process rather than handled by an external, global system (e.g., by some exception handling mechanism), designed, and added to the model describing the system's normal behavior. Thus, we are in favor of an internal and incremental description of changes over an external and uniform one and a local handling of changes over a global one.

A preliminary version of this work appeared in the *Proceedings of PDPTA '03* [3] in which we introduce the definitions of net rewriting systems and reconfigurable Petri nets. There are no proofs inside. This paper is an extended and improved version of [3]. Reconfigurable Petri nets are an extension of Petri nets [19], [20]. They are a subclass of net rewriting systems. The proposed model arises from two different lines of research. Both were conducted in the field of the Petri net formalism with the goal of enhancing the expressiveness of the basic model of Petri nets. The first class of models covers various proposals for merging Petri nets with graph grammars [7], [9], [21], while the second class, which is best represented by Valk's Self-Modifying Nets [23], [24], considers Petri nets whose flow relations can vary at runtime. Both proposals lead to expressive models that have definite benefits with respect to modeling issues. However, most of the basic decidable properties of Petri nets (place boundedness, reachability, deadlock, and liveness) are lost for these extended models. Therefore, no automatic verification tools can be implemented in the context of these models. Reconfigurable Petri nets attempt to combine the most relevant aspects of both of these approaches and constitute a class of models for which each of the preceding fundamental properties are decidable. This model should then be amenable to automatic verification.

In Section 2, we recall the definition of Net Rewriting System of [3]. Section 3 develops a simulation of a Turing Machine using Net Rewriting Systems. In Section 4, we present our model of reconfigurable Petri nets and we show an example of this model. An implementation of reconfigurable Petri nets with Petri nets is shown in Section 5. Finally, we conclude in Section 6 with some related work.

2 Net Rewriting Systems

This section recalls the model of the *net rewriting system* of [3]. First, we establish some notations:

• *The authors are with the Departamento de Sistemas Informáticos y Computación (DSIC), Universidad Politécnico de Valencia (UPV), Camino de Vera, s/n E-46022 Valencia, Spain.*
E-mail: {mllorens, fjoliver}@dcis.upv.es.

Manuscript received 18 June 2003; revised 9 Jan. 2004; accepted 20 Jan. 2004. For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0064-0603.

- If $R \subseteq X \times Y$ is a binary relation, we let $X'R = \{y \in Y \mid \exists x \in X'\ (x,y) \in R\}$ denote the *image* of $X' \subseteq X$, and $RY' = \{x \in X \mid \exists y \in Y'\ (x,y) \in R\}$ denote the *inverse image* of $Y' \subseteq Y$. The *domain* of R is then $Dom(R) = RY$ and the *codomain* of R is $Cod(R) = XR$.
- $\mathbb{N}$ is the set of natural numbers.
- $\mathbb{Z}$ is the set of integer numbers.

Definition 1. *A* Petri net *[19], [20] is a tuple $\mathcal{N} = (P, T, F)$, where $P = \{p_1, p_2, \ldots, p_m\}$ is a finite set of places, $T = \{t_1, t_2, \ldots, t_n\}$ is a finite set of transitions ($P \cap T = \emptyset$, $P \cup T \neq \emptyset$), and $F : (P \times T) \cup (T \times P) \to \mathbb{N}$ is a set of weighted arcs (flow relation).*

Definition 2. *A* marking of a Petri net *is a map $M : P \to \mathbb{N}$ that assigns, to each place, a nonnegative integer. If a marking assigns to place p a nonnegative integer k, we say that p* is marked with k tokens*. A marking is denoted by M, an m-vector, where m is the total number of places. The pth component of M, denoted by $M(p)$, is the number of tokens in place p.*

Definition 3. *A* marked Petri net *is a pair $(\mathcal{N}, M_0)$, where $\mathcal{N}$ is a Petri net and $M_0 : P \to \mathbb{N}$ is the initial marking.*

A Petri net structure can be represented as a directed graph that uses circles and bars to represent places and transitions, respectively. Arcs are labeled with their weights (positive integers), where a k-weighted arc can be interpreted as the set of k parallel arcs. Labels for unity weight are usually omitted. An arc from a place p to a transition t defines p as an *input place* of t. An arc from a transition t to a place p defines p as an *output place* of t. Tokens are represented graphically by black dots. If a place p is marked with k tokens, we place k black dots in place p.

Definition 4. *The* preset *(*postset*) of a transition t is the set of all input (output) places of t, ${}^\bullet t = \{\forall p \in P \mid F(p,t) \neq 0\}$ ($t^\bullet = \{\forall p \in P \mid F(t,p) \neq 0\}$).*

Definition 5. *The* preset *(*postset*) of a place p is the set of all input (output) transitions of p, ${}^\bullet p = \{\forall t \in T \mid F(t,p) \neq 0\}$ ($p^\bullet = \{\forall t \in T \mid F(p,t) \neq 0\}$).*

The *execution* of a Petri net causes its marking to change. Execution is performed by *firing* enabled transitions. A transition is *enabled* when each one of its input places is marked with at least as many tokens as the weight of the arcs connecting these input places with the transition. A transition fires by removing as many tokens from each one of its input places as the weight of the arcs connecting these input places with the transition and by placing as many tokens in each one of its output places as the weight of the arcs connecting these output places with the transition.

Definition 6. *A transition is* enabled *if each one of its input places is marked with at least $F(p,t)$ tokens, where $F(p,t)$ is the weight of the arc from p to t.*

$$\forall p \in {}^\bullet t \quad M(p) \geq F(p,t).$$

Definition 7. *A transition can be* fired *if and only if it is enabled.*

Definition 8. *The* firing *of an enabled transition t in a marking M removes $F(p,t)$ tokens from each input place p of t and it adds $F(t,p)$ tokens to each output place p of t, where $F(t,p)$ is the weight of the arc from t to p.*

Definition 9. *The marking M' resulting from the firing of an enabled transition t in a marking M, $M[t\rangle M'$, is defined as follows:*

$$\begin{array}{ll} M'(p) = M(p) - F(p,t) + F(p,t) & \forall p \in ({}^\bullet t \cap t^\bullet) \\ M'(p) = M(p) - F(p,t) & \forall p \in {}^\bullet t \backslash t^\bullet \\ M'(p) = M(p) + F(t,p) & \forall p \in t^\bullet \backslash {}^\bullet t \\ M'(p) = M(p) & otherwise. \end{array}$$

Definition 10. *The* marking graph *of a marked Petri net $(\mathcal{N}, M_0)$ is a graph G whose nodes are reachable markings from M_0 and whose arcs are firings of transitions. There exists an arc, labeled with t, from the node representing marking M_i to the node representing marking M_j if and only if, when t is fired from M_i, M_j is reached.*

Definition 11. *Let $G = (V, E)$ and $G' = (V', E')$ be two graphs. We call G and G'* isomorphic *if there exists a bijection $\varphi : V \to V'$ with $xy \in E \Leftrightarrow \varphi(x)\varphi(y) \in E'$ for all $x, y \in V$.*

Definition 12. *Let $\mathcal{N} = (P, T, F)$ and $\mathcal{N}' = (P', T', F')$ be two Petri nets. We call $\mathcal{N}$ and $\mathcal{N}'$* isomorphic *if there exists a bijection $\varphi : (P \cup T) \to (P' \cup T')$ such that $F(x,y) \in \mathcal{N} = F'(\varphi(x), \varphi(y)) \in \mathcal{N}'$ for all $x, y \in P \cup T$.*

Definition 13. *A* full embedding *of a Petri net $\mathcal{N} = (P, T, F)$ into a Petri net $\mathcal{N}' = (P', T', F')$ is an* injective map *$f : P \cup T \to P' \cup T'$ that maps places to places and transitions to transitions ($f(P) \subseteq P'$ and $f(T) \subseteq T'$) such that, for any pair of elements $x, y \in P \cup T$, $F(x,y) = F'(f(x), f(y))$. The image of $\mathcal{N}$ by f is then called a* full subnet *of $\mathcal{N}'$.*

We show the definition of the net rewriting system. This definition combines Petri nets with graph rewriting systems. The idea is to describe a system configuration as a Petri net (like in self-modifying nets [23], [24]) and a change of configuration as a graph rewriting rule (like a production and a direct derivation of a graph grammar in the double-pushout approach [11]).

Definition 14. *A* net rewriting system *[3] is a structure $N = (\mathcal{R}, (\Gamma_0, M_0))$, where $\mathcal{R} = \{r_1, \ldots, r_h\}$ is a finite set of rewriting rules and (Γ_0, M_0) is a marked Petri net.*

A rewriting rule *$r \in \mathcal{R}$ is a structure $r = (L, R, \tau, {}^\bullet\tau, \tau^\bullet)$, where*

1. *$L = (P_L, T_L, F_L)$ and $R = (P_R, T_R, F_R)$ are Petri nets called the* left-hand side *and the* right-hand side *of r, respectively.*
2. *$\tau \subseteq (P_L \times P_R) \cup (T_L \times T_R)$, called the* transfer relation *of r, is a binary relation relating places of L to places of R and transitions of L to transitions of R: $P_L\tau \subseteq P_R$, $\tau P_R \subseteq P_L$, $T_L\tau \subseteq T_R$, and $\tau T_R \subseteq T_L$.*
3. *${}^\bullet\tau \subseteq \tau$ and $\tau^\bullet \subseteq \tau$ are subrelations of the transfer relation called the* input interface *relation and the* output interface *relation, respectively.*

A configuration *of a net rewriting system N is a Petri net $\Gamma = (P, T, F)$.*

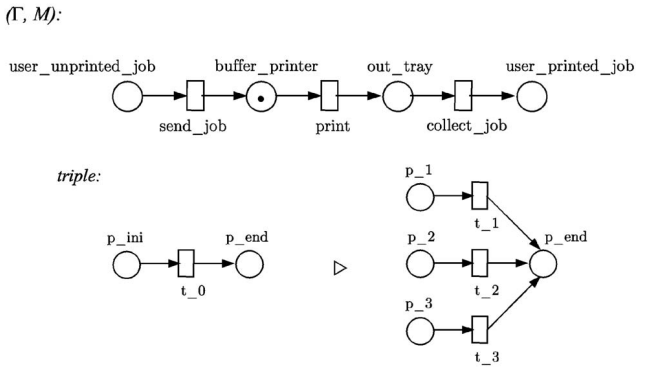

Fig. 1. Net Rewriting System modeling a system of printers.

A state *of a net rewriting system N is a marked Petri net (Γ, M). The pair (Γ_0, M_0) is called the* initial state *of the net rewriting system.*

An event *of a net rewriting system is either a transition or a rewriting rule: $E = T \cup R$.*

Let us present an example to illustrate this definition.

Example 1 (Printers). The net rewriting system in Fig. 1 models the printing of several copies of the same job using different printers. The token in the place *buffer_printer* represents one copy of the job to print. In this state (Γ, M), to obtain three copies the job must be sent to print three times and these three copies can only be printed sequentially (one after the other). The rewriting rule *triple* offers the possibility of printing each one of them using a different printer. The transfer relation τ is given by

$$\tau = \{(\{p\_ini\}, \{p\_1, p\_2, p\_3\}), (\{t\_0\}, \{t\_1, t\_2, t\_3\}), (\{p\_end\}, \{p\_end\})\}$$

and the input and output interface relations are ${}^\bullet\tau = \{(\{p\_ini\}, \{p\_1, p\_2, p\_3\})\}$ and $\tau^\bullet = \{(\{p\_end\}, \{p\_end\})\}$, respectively.

In order to apply a rewriting rule r to a configuration Γ, one must first identify a full subnet Γ' of Γ (see Definition 13) that is isomorphic to the left-hand side of the rule, that is, there exists a bijection $\varphi : (P_L \cup T_L) \to (P' \cup T')$ such that $F_L(x, y) \in L = F'(\varphi(x), \varphi(y)) \in \Gamma'$ for all $x, y \in P_L \cup T_L$. The elements of Γ (places or transitions) that do not belong to Γ' constitute the *context* of the rule. In order for the rule to be enabled, an element x' of Γ' must also have an element x of its preset that belongs to the context only if x' belongs to the input interface of the rule. In addition, an element x' of Γ' must also have an element x of its postset that belongs to the context only if x' belongs to the output interface of the rule. When these conditions are met, the rewriting can take place and it proceeds by replacing the subnet Γ' with the right-hand side R of the rule and by fixing the connections between the elements of R and those in the context according to the interface relation. The transfer relation is not only used to rewrite the left side of the rule with the right side, but it is also used to transfer the tokens in Γ' to R (hence its name). Notice that, since the transfer relation can be any type of relation, tokens may be duplicated or may disappear.

The dynamic evolution of a net rewriting system is given by its state graph.

Definition 15. *The* state graph of a net rewriting system $N = (\mathcal{R}, (\Gamma_0, M_0))$ *is the labeled directed graph whose nodes are the states of N, i.e., marked Petri nets, and whose arcs (labeled with events of N) are of two kinds described below:*

- Firing of a transition: *Arcs from state (Γ, M) to state (Γ', M') that are labeled with transition t when transition t can fire in the net Γ at marking M and lead to marking M':*

  $$(\Gamma, M) \xrightarrow{t} (\Gamma', M') \Longleftrightarrow (\Gamma = \Gamma' \text{ and } M[t\rangle M' \text{ in } \Gamma).$$

- Change of configuration: *Arcs from state (Γ, M) to state (Γ', M') that are labeled with rule $r = (L, R, \tau, {}^\bullet\tau, \tau^\bullet) \in \mathcal{R}$, when there exists a* full embedding *$f : L \to \Gamma$ such that, for all $x \notin f(L)$ and $y \in L$:*

  $$x \in {}^\bullet f(y) \Rightarrow y \in Dom({}^\bullet\tau) \text{ and } x \in f(y)^\bullet \Rightarrow y \in Dom(\tau^\bullet)$$

  and the following holds where $\Gamma = (P, T, F)$ and $\Gamma' = (P', T', F')$:

  $$P' = P - f(P_L) + P_R \text{ such that } P_L\tau = P_R$$
  $$T' = T - f(T_L) + T_R \text{ such that } T_L\tau = T_R,$$

  where the meaning of $+$ $(-)$ is adding (removing) places/transitions to (from) Γ. The name of places P_R (transitions T_R) added to Γ must be new in order to avoid clashes.

  The flow relation F' is given by:

  - *arcs in Γ that connect pairs of elements that do not belong to R,*
  - *arcs that connect pairs of elements of R, and*
  - *arcs that are the result of connecting elements of R with elements that belong to the context,*

  i.e.,

  $$F'(x, y) = \begin{cases} F(x, y) & if\ x \notin R \wedge y \notin R \\ F_R(x, y) & if\ x \in R \wedge y \in R \\ \sum_{y_i \in {}^\bullet\tau y} F(x, f(y_i)) & if\ x \notin R \wedge y \in R \\ \sum_{x_i \in \tau^\bullet x} F(f(x_i), y) & if\ x \in R \wedge y \notin R. \end{cases} \quad (1)$$

  The marking of a place $p \in P'$, $M'(p)$, is given by:

  - *If $p \notin R$, it does not change or*
  - *If $p \in R$, it depends on markings of the places from which is originated according to τ,*

  i.e.,

  $$M'(p) = \begin{cases} M(p) & if\ p \notin R \\ \sum_{p' \in \tau p} M(f(p')) & if\ p \in R. \end{cases} \quad (2)$$

Example 2. Fig. 2 shows the new state (Γ', M') due to the change of configuration caused by rewriting rule *triple* in Example 1. It could be appreciated that a subnet has been

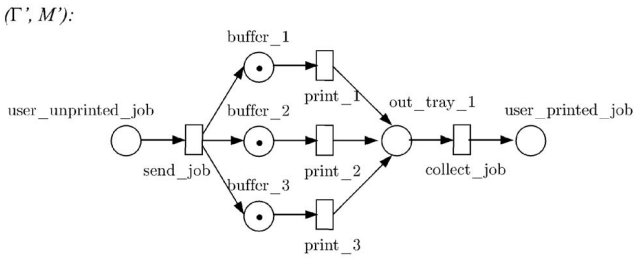

Fig. 2. Change of configuration due to rewriting rule tuple.

replaced by another one and the number of tokens has grown (it has been tripled).

In the next section we prove that net rewriting systems can simulate Turing machines.

3 Simulation of Turing Machines with Net Rewriting Systems

In this section, we show how to simulate a Turing machine by means of a net rewriting system. First, we recall the elements that make up a Turing machine, the definition of a Turing machine configuration and the transition relation between configurations.

Definition 16. *A* Turing machine *[16], [22] is made up of the following elements:*

- *an infinite bidirectional tape divided into an infinite number of consecutive cells, every cell contains* 0 *or* 1,
- *a read/write head that can read and write into the active cell (the cell where is the head) and that can also move along the tape in both directions,*
- *a finite set of internal states* Q, *and*
- *a finite set of instructions of the form* $[q, v, w, q']$ *where* q *and* q' *are internal states,* $v \in \{0, 1\}$ *is a possible value of a cell, and* $w \in \{0, 1, L, R\}$ *is an operation. The meaning of this instruction is: If the internal state of the machine is* q *and the content of the cell being examined is the symbol* v, *then the machine performs the operation* w *and changes its internal state to* q'. *According to the operation* w, *the read/write head can:*
  - *write* 0 (1) *in the cell being examined if* $w = 0$ $(w = 1)$ *and*
  - *move to the next cell to the left or to the right if* $w = L$ *or* $w = R$.

Definition 17. *A* configuration *of a Turing machine is a pair* $(q, u) \in Q \times \{0, 1\}^{\mathbb{Z}}$ *where* $q \in Q$ *is a state and* $u : \mathbb{Z} \to \{0, 1\}$ *is a function describing the current status of the tape (*0 *is the head position). The initial configuration is* (q_0, u_0) *where* $u_0(n) = 0$ *for all* $n \in \mathbb{Z}$, *i.e., initially every cell has value* 0.

Definition 18. *The* transition relation *between configurations of a Turing machine is given by:*

$$(q, u) \xrightarrow{x} (q', u') \Longleftrightarrow \begin{cases} (x = [q, u(0), u'(0), q'] \wedge \forall i \neq 0\ u'(i) = u(i)) \vee \\ (x = [q, u(0), R, q'] \wedge \forall i \in \mathbb{Z}\ u'(i) = u(i+1)) \vee \\ (x = [q, u(0), L, q'] \wedge \forall i \in \mathbb{Z}\ u'(i) = u(i-1)). \end{cases}$$

A Turing machine is *deterministic* if there are not two distinct rules $[q_1, v_1, w_1, q_1']$ and $[q_2, v_2, w_2, q_2']$ with $q_1 = q_2$ and $v_1 = v_2$. This means that, in any configuration of the machine, at most one instruction may apply. The behavior of a deterministic Turing machine can be represented by the sequence of configurations reached from its initial configuration. Thus, a Turing machine may either halt, if it reaches a configuration where no instructions apply, or cycle, if it reaches some configuration twice, or be unbounded, if it reaches infinitely many distinct configurations. It is undecidable whether a Turing machine halts and it is also undecidable whether a Turing machine is bounded. It may be proven that a Turing machine is bounded if and only if its head scans only a finite part of the tape. Any configuration reachable from the initial configuration has only a finite number of cells with a nonnull content. Therefore, at any time, only a finite portion of the tape needs to be represented.

Fig. 3 shows the equivalence between a configuration of a Turing machine and a configuration of a net rewriting system. In this representation, a cell of the machine is a place with a label 0 or 1, that we call *sort*, and that represents its content. Each state of the Turing machine is also considered a sort and the active cell is represented by the place that has an input transition with the current state of the machine as sort. We represent the fact that a cell is the leftmost cell, an internal, or the rightmost cell with specific sorts. The transition relations between configurations of the machine are rewriting rules of the net rewriting system.

Next, we show that the model of net rewriting systems is left unchanged if we add sort constraints on places and transitions. We introduce the notion of *K*-sorted net rewriting system as an extension of net rewriting systems with some sort constraints on places and transitions. We use these net rewriting systems to simulate a Turing machine.

Definition 19. *A net* $\Gamma = (P, T, F)$ *is the* restriction *of a net* $\Gamma' = (P', T', F')$ *if* $P \subseteq P'$, $T \subseteq T'$, *and* $F = F' \cap (P \times T \cup T \times P)$. *It is denoted by* $\Gamma \sqsubseteq \Gamma'$.

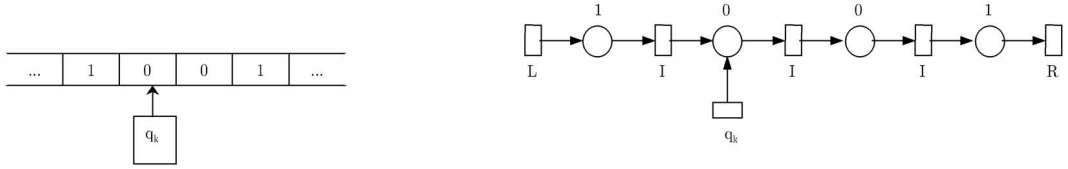

Fig. 3. Implementation of a Turing Machine with a net rewriting system.

Definition 20. *A net rewriting system $N = (\mathcal{R}, (\Gamma_0, M_0))$ is the* restriction of a net rewriting system $N' = (\mathcal{R}', (\Gamma'_0, M'_0))$ *if:*

1. $\Gamma_0 \sqsubseteq \Gamma'_0$ *and*
2. $\exists$ *a bijection* $\varphi : \mathcal{R} \to \mathcal{R}'$ *such that* $\forall r \in \mathcal{R}$, $L_r \sqsubseteq L'_r$ *and* $R_r \sqsubseteq R'_r$;

it is denoted by $N \sqsubseteq N'$.

Definition 21. *The* restriction of the state graph *of a net rewriting system N' to the state graph of a net rewriting system N if $N \sqsubseteq N'$, is the transition system $(\Gamma, M) \xrightarrow{x} (\Gamma', M')$ such that*

- Γ *and* Γ' *are configurations of* N,
- $x \in T \cup \mathcal{R}$,
- M *and* M' *are the markings of* Γ *and* Γ', *respectively, and*
- $\exists(\tilde{\Gamma}, \tilde{M}) \xrightarrow{\tilde{x}} (\tilde{\Gamma}', \tilde{M}')$ *is a transition of the state graph of* N' *such that:*

  -
  $$\tilde{x} = \begin{cases} x & if\ x \in T_\Gamma \\ \varphi_x & if\ x \in \mathcal{R}, \end{cases}$$

  - $\Gamma \sqsubseteq \tilde{\Gamma}$ *and* $\Gamma' \sqsubseteq \tilde{\Gamma}'$,
  - $M = \tilde{M}$ *restricted to places of* Γ *and* $M' = \tilde{M}'$ *restricted to places of* Γ'.

Definition 22. *A net rewriting system N' is a* conservative extension *of a net rewriting system N if $N \sqsubseteq N'$ and the state graph of N coincide (is isomorphic) with the restriction of the state graph of N' to the state graph of N.*

Definition 23. *If K is some finite set (of so-called* sorts*), a K-*sorted Petri net *is a Petri net (P, T, F) together with a map $\kappa : P \cup T \to K$ that associates each element of the net with a sort in K.*

Definition 24. *A K-*sorted net rewriting system *is a net rewriting system all of whose Petri nets components (initial configuration and left and right-hand side of rules) are K-sorted.*

Definition 25. *The* state graph of a *K-*sorted net rewriting system *is defined as the state graph of its underlying net rewriting system with the exception that, for applying a change of configuration, we further require that the embedding (of the left-hand side of the rule into the current configuration) be compatible with the sorts, i.e., it sends an element (place or transition) to an element with the same sort.*

Proposition 1. *For every K-sorted net rewriting system N, there exists a net rewriting system N' which is a conservative extension of the underlying net rewriting system N_0 to N such that the restriction of the state graph of N' to the state graph of N_0 coincides with the state graph of N.*

Proof.

- By the fact that N_0 is the underlying net rewriting system (without sorts) to the K-sorted net rewriting system N, it follows that $N_0 \sqsubseteq N$, i.e., N_0 is the restriction of N.
- By the fact that N' is a conservative extension of system N_0, it follows that $N_0 \sqsubseteq N'$ and the state graph of N_0 coincides with the restriction of state graph of N' to the state graph of N_0.
- From Definition 25, the state graph of N is defined as the state graph of N_0 with the exception that, for applying a change of configuration, we further require that the embedding (of the left-hand side of the rule into the current configuration) be compatible with the sorts, i.e., the state graph of N coincides with the state graph of N_0.

Therefore, the restriction of the state graph of N' to the state graph of N_0 coincides with the state graph of N. □

According to Proposition 1, we deduce that:

Proposition 2. *Let N_0 be the underlying net rewriting system to the K-sorted net rewriting system N. If the state graph of N_0 coincides with the state graph of N, places and transitions of the sort set are unobservables, i.e., they have no effect in the state graph of N.*

Proof. It follows from Proposition 1. □

Next, we show that if we relax slightly the assumption on the transfer relations in a net rewriting system, now requiring that they are *partial bijections*, then we obtain a larger subclass of net rewriting systems that already is Turing powerful. We recall the notion of partial bijection.

Definition 26. *A relation $\rho \subseteq X \times Y$ is a* partial bijection *if it induces a bijection when restricted to its domain and its codomain, i.e., each element in X has at most one image in Y and each element in Y has at most one coimage in X.*

The fact that Turing machines can be simulated by net rewriting systems whose transfer relations are partial bijections is straightforward when one has noticed that the model of net rewriting systems is left unchanged if we add sort constraints on places and transitions.

Now, we readily verify that we can associate every K-sorted net rewriting system whose transfer relations are partial bijections to an equivalent net rewriting system whose transfer relations are also partial bijections. For that purpose, we notice that, since the transfer relations are partial bijections, the flow relations of its various configurations are upper bounded by the maximum of the values of the flow relations of the initial configuration and of the values of the flow relations of the right-hand sides of rules. Then, a sort n of a transition t and a sort m of a place p may be coded by the specific subnets, added to the original nets (see Fig. 4), once one has chosen these integers large enough so that they cannot conflict with other values of flow relations of some accessible configuration.

We show below that any Turing machine can be simulated by some sorted net rewriting system whose transfer relations are partial bijections.

Theorem 1. *A K-sorted net rewriting system whose transfer relation τ is a partial bijection is Turing powerful.*

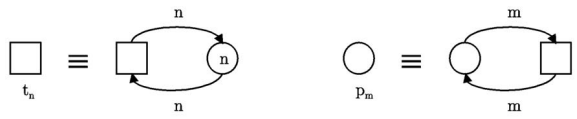

Fig. 4. Subnets of a sort n of a transition t and a sort m of a place p.

Proof. Formally, a Turing machine is represented by a K-sorted net rewriting system as follows: Let $N = (\mathcal{R}, (\Gamma_0, M_0))$ be the K-sorted net rewriting system where

- $K = \{0,1\} \cup Q \cup \{L, R, I\}$ is the sort set such that $\{0,1\}$ are sorts of places and indicate the content of the cell represented by the corresponding place and transitions can have sorts in $Q \cup \{L, R, I\}$, where
  - Q is a finite set of sorts coding the various internal states of the machine. The current configuration state q_k is represented by an input transition to the place of the active cell with sort $q_k \in Q$;
  - L (R) is the sort of the input (output) transition of the place that represents the leftmost (rightmost) cell of the tape, and
  - I is the sort of all transitions between places representing internal cells.
- $\Gamma_0 = (P_0, T_0, F_0)$ is the initial configuration, such that $P_0 = \{p_0\}$, $T_0 = \{q_0, L, R\}$ and F_0 is defined as follows:

$$F_0(p_0, q_0) = 0, F_0(q_0, p_0) = 1, F_0(p_0, L) = 0,$$
$$F_0(L, p_0) = 1, F_0(p_0, R) = 1, \text{and } F_0(R, p_0) = 0.$$

  This is illustrated graphically in Fig. 5.
- M_0 is the initial marking such that $M_0(p_0) = 0$,
- $\mathcal{R}$ is the set of rewriting rules and represents the transition relations between configurations of the Turing machine. The rules are the following:
  - *Writing a value on the cell being examined*. There is one rewriting rule associated with each instruction of the Turing machine of the form $[q, v, v', q'] \in Q \times \{0,1\}^2 \times Q$ (see Fig. 6).

    The transfer relation τ is given by $\tau = \{(\{v\}, \{v'\}), (\{q\}, \{q'\})\}$ and the input and output interface relations are ${}^\bullet\tau = \tau^\bullet = \{(\{v\}, \{v'\})\}$.
  - *Moving the head to the right*. There are two rewriting rules associated with each instruction of the Turing machine of the form $[q, v, R, q']$ according to whether the active cell is the rightmost cell (see Fig. 7a) or not (see Fig. 7b).

    The transfer relation τ for the rule of Fig. 7a is given by $\tau = \{(\{v\}, \{v\}), (\{R\}, \{R\}), (\{q\}, \{q'\}), (\emptyset, \{I, 0\})\}$ and the input and output interface relations are ${}^\bullet\tau = \{(\{v\}, \{v\})\}$ and $\tau^\bullet = \{\emptyset\}$, respectively.

    The transfer relation τ for the rule of Fig. 7b is given by $\tau = \{(\{v\}, \{v\}), (\{I\}, \{I\}), (\{v'\}, \{v'\}), (\{q\}, \{q'\})\}$ and the input and output interface relations are ${}^\bullet\tau = \{(\{v\}, \{v\})\}$ and $\tau^\bullet = \{(\{v'\}, \{v'\})\}$, respectively.
  - *Moving the head to the left*. There are two rewriting rules associated with each instruction of the Turing machine of the form $[q, v, L, q']$ according to whether the active cell is the leftmost cell (see Fig. 8a) or not (see Fig. 8b).

    The transfer relation τ for the rule of Fig. 8a is given by $\tau = \{(\{v\}, \{v\}), (\{L\}, \{L\}), (\{q\}, \{q'\}), (\emptyset, \{0, I\})\}$ and the input and output interface relations are ${}^\bullet\tau = \{\emptyset\}$ and $\tau^\bullet = \{(\{v\}, \{v\})\}$, respectively.

    The transfer relation τ for the rule of Fig. 8b is given by $\tau = \{(\{v\}, \{v\}), (\{I\}, \{I\}), (\{v'\}, \{v'\}), (\{q\}, \{q'\})\}$ and the input and output interface relations are ${}^\bullet\tau = \{(\{v\}, \{v\})\}$ and $\tau^\bullet = \{(\{v'\}, \{v'\})\}$, respectively. □

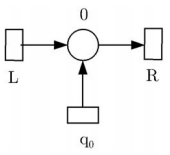

Fig. 5. Net of the initial configuration.

Fig. 6. Writing a value on the cell being examined.

This translation show how simulate a deterministic Turing machine with a K-sorted net rewriting system where τ is a partial bijection.

This simulation can be easily extended to the general class of net rewriting systems since we have proven that the model of net rewriting systems is left unchanged if we add sort constraints on places and transitions. This is correct only if concurrency aspects are not taken into account. We can conclude that:

Corollary 1. *Net rewriting systems are Turing powerful.*

Proof. Straightforward. □

In the next section, we present a particular class of net rewriting system, the reconfigurable Petri nets, where the transfer relation is a bijection. In Section 5, we show that this model is an abbreviation of the basic model of Petri nets, i.e., having the same expressive power, the description of the system to model is simpler and brief.

4 Reconfigurable Petri Nets

We now turn our attention to a specific subclass of net rewriting systems corresponding to a model of reconfigurable Petri nets that we have introduced in previous studies

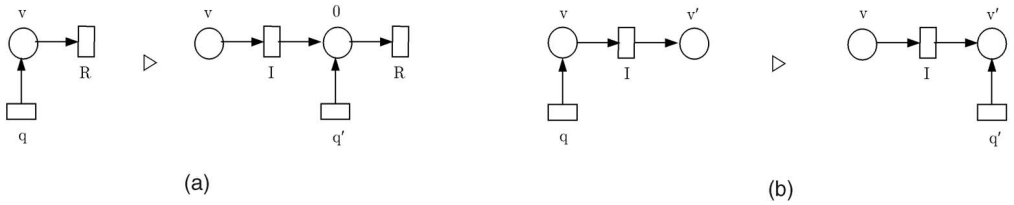

Fig. 7. Moving the head to the right. (a) The active cell is the rightmost cell. (b) The active cell is not the rightmost cell.

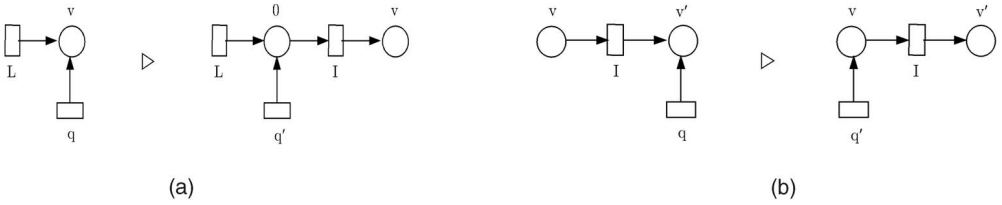

Fig. 8. Moving the head to the left. (a) The active cell is the leftmost cell. (b) The active cell is not the leftmost cell.

[4], [5], [6]. More precisely, the following definition is a reformulation of what was called there *reversible* reconfigurable nets.

Definition 27. *A* reconfigurable Petri net *is a net rewriting system such that the transfer relation τ is a bijection and the interfaces $\tau^\bullet = {}^\bullet\tau$ are the restriction of the transfer relation τ to the set of transitions, i.e., $\tau^\bullet = {}^\bullet\tau = T_L \times T_R \cap \tau$.*

The former condition asserts that the set of places and transitions is left unchanged by rewriting rules. Such rules only change the flow relations of the places in their domains. Thus, the above definition may in turn be reformulated more directly (i.e., without resorting to net rewriting systems) using the following definition, which we consider the official definition of reconfigurable Petri nets (since reversibility will always be an implicit assumption).

Definition 28. *A* reconfigurable Petri net *is a structure $N = (P, T, R, \gamma_0)$ where $P = \{p_1, \ldots, p_n\}$ is a nonempty and finite set of places, $T = \{t_1, \ldots, t_m\}$ is a nonempty and finite set of transitions disjoint from P ($P \cap T = \emptyset$), $R = \{r_1, \ldots, r_h\}$ is a finite set of rewriting rules, and γ_0 is the initial state.*

A rewriting rule *$r \in R$ is a structure $r = (D, {}^\bullet r, r^\bullet)$ where $D \subseteq P$ is the* domain *of r, ${}^\bullet r : (D \times T) \cup (T \times D) \to \mathbb{N}$ and $r^\bullet : (D \times T) \cup (T \times D) \to \mathbb{N}$ are the* preconditions *and* postconditions *of r (i.e., they are the flow relations of the domain places before and after the change of configuration due to rule r).*

A configuration *of a reconfigurable Petri net is a Petri net $\Gamma = (P, T, F)$.*

A state *γ of a reconfigurable Petri net is a pair $\gamma = (\Gamma, M)$ where Γ is a configuration and $M : P \to \mathbb{N}$. Hence, a state is a marked Petri net that is associated with the set of places and transitions of the reconfigurable Petri net.*

The events *of a reconfigurable Petri net are its transitions together with its rewriting rules: $E = T \cup R$.*

We represent a rewriting rule using formal sums notation as:

$$r = \sum_{p \in D} p\left(\sum_{t \in T} {}^\bullet r(p,t) \cdot t - \sum_{t \in T} {}^\bullet r(t,p) \cdot t\right) \triangleright \sum_{p \in D} p\left(\sum_{t \in T} r^\bullet(p,t) \cdot t - \sum_{t \in T} r^\bullet(t,p) \cdot t\right),$$

where the symbol $\triangleright$ separates the preconditions from the postconditions of r.

Definition 29. *The* configuration graph *$G(N)$ of a reconfigurable Petri net $N = (P, T, R, \gamma_0)$ is the labeled directed graph whose nodes are the configurations such that there is an arc from configuration Γ to configuration Γ' labeled with rule $r = (D, {}^\bullet r, r^\bullet) \in R$, which we denote $\Gamma[r\rangle\Gamma'$, if and only if the following holds:*

$$\begin{aligned} &\forall p \in D : \begin{cases} F(p,t) = {}^\bullet r(p,t) \text{ and } F(t,p) = {}^\bullet r(t,p) \\ F'(p,t) = r^\bullet(p,t) \text{ and } F'(t,p) = r^\bullet(t,p) \end{cases} \\ &\forall p \notin D : F(p,t) = F'(p,t) \text{ and } F(t,p) = F'(t,p). \end{aligned} \tag{3}$$

Notice that we want the transition relation to contain arcs of the exact multiplicity appearing in the the left-hand side of the rewriting rule and we do not allow rewriting if arcs of a greater multiplicity are present.

The dynamic evolution of a reconfigurable Petri net is then given by its state graph.

Definition 30. *The* state graph *of a reconfigurable Petri net $N = (P, T, R, \gamma_0)$ is the labeled directed graph whose nodes are states of N and whose arcs (labeled with events) are of two kinds:*

- Firing of a transition: *Arcs from state (Γ, M) to (Γ, M') that are labeled with transition t when transition t can fire in the net Γ at marking M and leads to marking M',*
- Change of configuration: *Arcs from state (Γ, M) to state (Γ', M) that are labeled with rule $r \in R$ if $\Gamma[r\rangle\Gamma'$ is a transition of the configuration graph of N.*

In other words, the set of labeled arcs of the state graph of N is given by

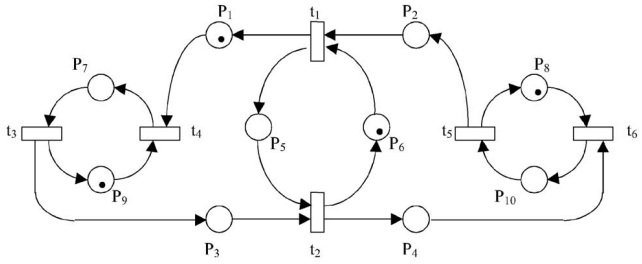

Fig. 9. Two half-duplex communicating lines between two producers/consumers.

$$\{(\Gamma, M) \xrightarrow{t} (\Gamma, M') \mid M[t\rangle M' \; in \; \Gamma\} \cup$$
$$\{(\Gamma, M) \xrightarrow{r} (\Gamma', M) \mid \Gamma[r\rangle \Gamma' \; in \; G(N)\}.$$

It is obvious from the previous definition that the evolution of a system modeled by a reconfigurable Petri net depends on what event takes place. If a transition is fired, only the net marking is affected (as in Petri nets) whereas, if a rewriting rule is applied, the net structure is modified and the marking continues being the original marking (i.e., changes of configurations are orthogonal to the behavior of the underlying net). On the other hand, in the general case of net rewriting systems, the application of a rewriting rule not only affects net structure but also net marking and, therefore, the behavior is distinct to the underlying net. Nevertheless, since there are only modeling and simulating tools for this case and it is not difficult to restrict the behavior so that changes of configurations take place when some system local properties occur, it is possible to take into account these limits at simulator level.

Each change of configuration that could take place in a reconfigurable Petri net is represented by a rewriting rule. To define these rules in a generic way, we can group places and transitions according to a designer decision. This doesn't change the nature of the considered model, but it facilitates its description. So, we let two places (transitions) play the same *role* if the designer groups them as the same kind of place (transition).

Example 3 (Producers/Consumers). Fig. 9 represents two half-duplex communicating lines between two producers/consumers. In this reconfigurable Petri net, some changes of configuration could take place. Each of them is represented by a rewriting rule. In the example, we distinguish the role P, Q, and S between places and the role A and B between transitions: $P = \{P_1, P_2, P_3, P_4\}$, $Q = \{P_5, P_6\}$, $S = \{P_7, P_8, P_9, P_{10}\}$, $A = \{t_1, t_2\}$, and $B = \{t_3, t_4, t_5, t_6\}$.

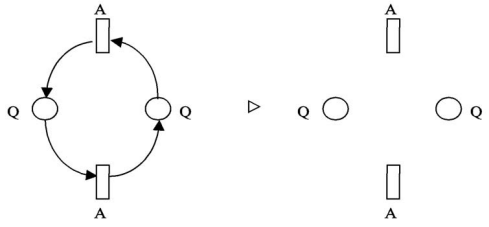

Fig. 10. Rewriting rule to change from sequential to parallel mode.

Fig. 11. Rewriting rule to change of sense in one communicating line.

We define four rewriting rules using the formal sums notation previously introduced.

1. Change from sequential to parallel mode (Fig. 10):

$$R_1 : Q(A - A) + Q(-A + A) \triangleright Q(\emptyset) + Q(\emptyset).$$

2. Change of sense in one communicating line (Fig. 11):

$$R_2 : P(-A) + P(A) \triangleright P(A) + P(-A).$$

3. Change from producer/consumer (Fig. 12):

$$R_3 : S(B - B) + P(-B) + S(-B + B) + P(B) \triangleright S(B - B) + P(B) + S(-B + B) + P(\emptyset).$$

4. Change from producer/consumer to producer (Fig. 13):

$$R_4 : S(B - B) + P(-B) + S(-B + B) + P(B) \triangleright S(B - B) + P(-B) + S(-B + B) + P(\emptyset).$$

If we change from sequential to parallel mode, we apply the rewriting rule R_1 and we obtain the state of Fig. 14 in which only the flow relations are changed (the marking is the original marking).

Fig. 15 shows part of the state graph of the reconfigurable Petri net example. We can appreciate that, depending on what event takes place, the net marking or the net topology is changed. When we apply a rewriting rule, we obtain a state in which only the flow relations are changed (the marking is the original marking), whereas, if a transition is fired, the marking changes and the structure is not modified.

5 Implementation of Reconfigurable Petri Nets with Petri Nets

The purpose of this section is to prove that reconfigurable Petri nets are equivalent to Petri nets. At first glance, it might seem they are not equivalent because of the set of

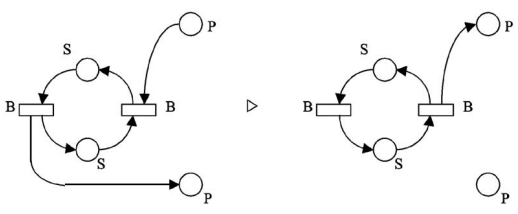

Fig. 12. Rewriting rule to change from producer/consumer to consumer.

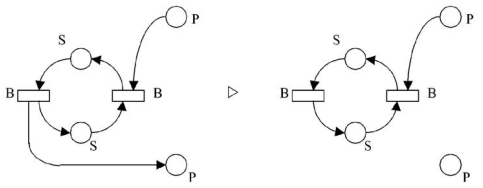

Fig. 13. Rewriting rule to change from producer/consumer to producer.

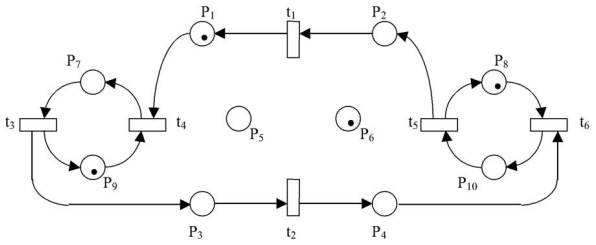

Fig. 14. Two half-duplex communicating lines between two producers/consumers.

rewriting rules of reconfigurable Petri nets. When a rewriting rule is applied in a reconfigurable Petri net, a change of configuration takes place (let $\Gamma_i[r\rangle\Gamma_j$ denote the change of configuration due to rewriting rule r from Γ_i to Γ_j) (i.e., a change in net structure). To obtain an equivalent Petri net, these changes of configuration must be present (i.e., all possible configurations must be represented in the net). We can therefore deduce that the number of configurations must be finite to be represented.

Let us call a *local configuration* a map $T \to \mathbb{N} \times \mathbb{N}$ that represents the incoming and outgoing flow relations with respect to the fixed set of transitions T of some place of a configuration of the reconfigurable Petri net. A configuration of a reconfigurable Petri net then amounts to associating each place (in the fixed set of places) with the corresponding local configuration. Now, the latter condition in the above definition shows that a local configuration of a reconfigurable Petri net should be either some local configuration of the initial configuration or some local configuration of a right-hand side of some rule. Thus, there are only finitely many local configurations and (since the set of places is itself fixed), only finitely many configurations. Let $Conf(N) = \{\Gamma_0, \Gamma_1, \ldots, \Gamma_k\}$ denote the set of configurations of a reconfigurable Petri net N, where Γ_0 is the configuration of the initial state $\gamma_0 = (\Gamma_0, M_0)$ (the initial configuration).

We can then easily construct an equivalent Petri net $\tilde{N} = (\tilde{P}, \tilde{T}, \tilde{F}, \tilde{M}_0)$ whose set of places is $\tilde{P} = P \cup \{q_0, \ldots, q_k\}$, (i.e., the places of the original reconfigurable Petri net together with one specific place associated with each possible configuration). We take as many copies of the set of transitions as the number of configurations, i.e., the set $\{q_0, \ldots, q_k\} \times T$. We can imagine the places and the transitions of a configuration $\Gamma_i = (P, T, F_i)$ as if they were located in two different parallel planes, i.e., a plane with the set of places and a plane with the set of transitions connected by flow relations of the represented configuration. Then, we set up flow relations so that the configuration Γ_i is represented by the plane of transitions $\{q_i\} \times T$ and the (shared) plane of places P:

$$\tilde{F}(p, (q_i, t)) = F_i(p, t) \text{ and } \tilde{F}((q_i, t), p) = F_i(t, p), \text{ where } \Gamma_i = (P, T, F_i).$$

Place q_i, which is associated with configuration Γ_i, contains at most one token and it is marked in the states that are associated with this configuration. Thus, we set

$$\tilde{F}(q_i, (q_j, t)) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

$$\tilde{F}((q_j, t), q_i) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

Then, it is only necessary to represent the change of configurations. For that purpose, it suffices to add one extra transition r_{ij} for each change of configuration $\Gamma_i[r\rangle\Gamma_j$ that has a flow relation given by $\tilde{F}(q_i, r_{ij}) = \tilde{F}(r_{ij}, q_j) = 1$ and $\tilde{F}(p, r_{ij}) = \tilde{F}(r_{ij}, p) = 0$, otherwise. So, the set of transitions is $\tilde{T} = (\{q_0, \ldots, q_k\} \times T) \cup \tilde{R}$, where $\tilde{R}$ is the set of transitions such that $r_{ij} \in \tilde{R}$ if $\exists r \in R$ such that $\Gamma_i[r\rangle\Gamma_j$ in $G(N)$. The resulting Petri net initially is marked as: $\tilde{M}_0(p) = M_0(p)$, where $p \in P$ and

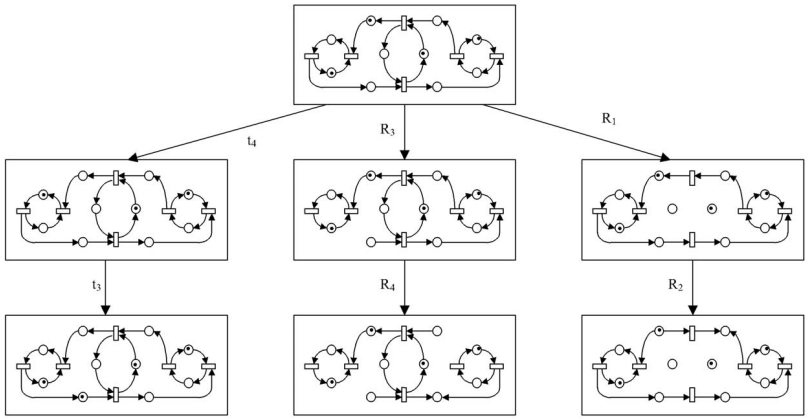

Fig. 15. (Part of) the state graph of the reconfigurable Petri net.

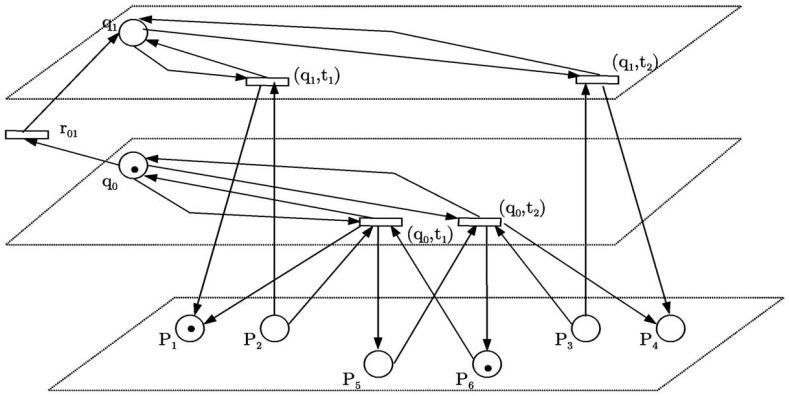

Fig. 16. Part of Petri net equivalent to Reconfigurable Petri net of Example 3.

$$\tilde{M}_0(q_i) = \begin{cases} 1 & \text{if } i = 0 \\ 0 & \text{otherwise.} \end{cases}$$

To show the correspondence between reconfigurable Petri nets and Petri nets some considerations might be taken into account:

- If $\tilde{M} : \tilde{P} \to \mathbb{N}$ is a reachable marking of $\tilde{N}$, then $\sum_{i=0}^{k} \tilde{M}(q_i) = 1$, that is, from the subset of places $\{q_0, \ldots, q_k\} \in \tilde{P}$, only one place is marked with one token, the place q_i which is associated with the current configuration Γ_i. From the subset of transitions $\{q_0, \ldots, q_k\} \times T \in \tilde{T}$, only the transitions in $\{q_i \times T\}$ were enabled.
- We denote $\gamma_{\tilde{M}} = (\Gamma_{\tilde{M}}, M)$, where $M : P \to \mathbb{N}$ is a marking of $\Gamma_{\tilde{M}}$ and $\Gamma_{\tilde{M}} = \Gamma_i$ is the configuration associated with $\tilde{M}$ such that $\tilde{M}(q_i) = 1$.
- Inversely, if $\gamma = (\Gamma, M)$ is a reachable state of N, we associate the mapping $\tilde{M}_\gamma : \tilde{P} \to \mathbb{N}$ with $\tilde{M}_\gamma(p) = M(p)$ if $p \in P$ and

$$\tilde{M}_\gamma(q_i) = \begin{cases} 1 & \text{if } \Gamma = \Gamma_i \\ 0 & \text{otherwise.} \end{cases}$$

- To abbreviate: $\tilde{M}_{(\gamma_{\tilde{M}})} = \tilde{M}$, $\gamma_{(\tilde{M}\gamma)} = \gamma, \gamma_{(\tilde{M}_0)} = (\Gamma_0, M_0)$ and $\tilde{M}_{\gamma_0} = \tilde{M}_0$.

Proposition 3. *Let N be a reconfigurable Petri net. If $\gamma = (\Gamma, M)$ is a reachable state of N, then*

1. $\gamma[r\rangle\gamma' \Longleftrightarrow \tilde{M}_\gamma[r\rangle\tilde{M}_{\gamma'}$, *where* $\gamma' = (\Gamma', M)$.
2. $\gamma[t\rangle\gamma' \Longleftrightarrow \tilde{M}_\gamma[(q_i, t)\rangle\tilde{M}_{\gamma'}$, *where* $\gamma = (\Gamma_i, M)$ *and* $\gamma' = (\Gamma_i, M')$.

Proof.

1. If we change from state $\gamma = (\Gamma_i, M)$ to state $\gamma' = (\Gamma_j, M)$ in a reconfigurable Petri net due to the firing of a rewriting rule, what differs in the equivalent Petri net is the marking of places q_i and q_j:

$$\tilde{M}_\gamma(q_i) = 1 \Longrightarrow \tilde{M}_{\gamma'}(q_i) = 0$$
$$\tilde{M}_\gamma(q_j) = 0 \Longrightarrow \tilde{M}_{\gamma'}(q_j) = 1.$$

   And also in the other direction.

2. If we change from state $\gamma = (\Gamma_i, M)$ to state $\gamma' = (\Gamma_i, M')$ in a reconfigurable Petri net due to the firing of a transition t, what changes in an equivalent Petri net is the marking of places $p \in P$ involved in the firing of transition (q_i, t) and vice versa. □

Hence, we have established that:

Theorem 2. *Let N be a reconfigurable Petri net and let $\tilde{N}$ be its equivalent Petri net. The markings $\tilde{M}_\gamma$ in which γ covers the reachable states of N are the reachable markings of $\tilde{N}$, and the marking graph of $\tilde{N}$ is isomorphic (see Definition 11) to the marking graph of N.*

Proof. The first part of this theorem follows from Proposition 3 and the second part is straightforward. □

In this sense,

Corollary 2. *Any reconfigurable Petri net is equivalent to some Petri net.*

Proof. Straightforward. □

Thus, reconfigurable Petri nets are equivalent to Petri nets, but they provide somewhat more compact representations of concurrent systems whose structures evolve at runtime. This can be observed in Fig. 16, which illustrates a fragment of the Petri net that is equivalent to the reconfigurable Petri net in Example 3. To facilitate the understanding of the implementation of a reconfigurable Petri net with a Petri net, we only show how to represent one change of configuration (the change from sequential to parallel mode) and only for the places and transitions involved. With this short fragment, we can imagine how big the entire Petri net is and how we can best model the system with a reconfigurable Petri net. In general, if the original reconfigurable Petri net $N = (P, T, \mathcal{R}, \gamma_0)$ has n places, m transitions, r rewriting rules, and the number of accesible configurations is $k+1$, the obtained equivalent Petri net $\tilde{N} = (\tilde{P}, \tilde{T}, \tilde{F}, \tilde{M}_0)$ consists of: $n + (k+1)$ places and $((k+1) * m) + z$ transitions, where $z = \sum_{i=0}^{k}(|\Gamma_i^\bullet| + |{}^\bullet\Gamma_i|)$ and $|\Gamma_i^\bullet|$ ($|{}^\bullet\Gamma_i|$) is the number of output (input) arcs from (to) the configuration Γ_i in the configuration graph of N, $G(N)$. That is, z is the number of transitions of $\tilde{R}$.

6 Related Work and Conclusions

We have introduced the net rewriting system, an extension of Petri nets suited for the modeling, simulation, and verification of concurrent systems subject to dynamic changes. Net rewriting systems are based on two different lines of research that extend the basic model of Petri nets, making possible the description of dynamic changes in concurrent systems: *graph grammars* [9], [21], [7] and *Valk's self-modifying nets* [23], [24]. The rewriting rules of net rewriting systems are very similar to productions of graph grammars (they have left and right-hand sides and interfaces) and the application of a rewriting rule is like a direct derivation in graph grammars (under certain conditions, once an occurrence (a *match*) of the left-hand side in a graph has been detected, it can be replaced by the right-hand side). As in self-modifying nets, we model a system which consists of a set of Petri nets, called *configurations*, and a mechanism that allows the system to evolve from one configuration to another under certain circumstances.

In the literature, besides Valk's self-modifying nets, there are some other models that allow for the description of complex dynamic concurrent systems: the *mobile nets*} of Asperti and Busi [1] originated from a merging of Petri nets with the name managing techniques typical in π-calculus [18] and the *dynamic nets* of Buscemi and Sassone [8] inspired by the join calculus [13], [14], that allow the dynamic creation of components as in our proposal; the Δ-$nets$ of Gradit et al. [15] and Vernadat et al. [25], a rewriting formalism which integrates the advantages of Petri nets and graph grammars, respectively, for behavior specification and topological transformations of a workflow; the *POP formalism* introduced by Engelfriet et al. [12] and the related model of *cooperating automata* by Badouel et al. [2], in which tokens are active elements with dynamic behavior. For all of them, like in our model, the description of changes is internal and incremental and their handling is local. Also, the idea of rewriting underlies all proposals, the configuration of the system is described as a Petri net, and a change of configuration as a graph rewriting rule which replaces the part of the system that matches the left-hand side of the rewriting rule by the corresponding right-hand side. With respect to the expressive power, all of them are Turing-equivalent, as is our model. The model of net rewriting systems is closer to Petri nets.

We have distinguished two particular classes of net rewriting systems according to their transfer relations τ. The model of reconfigurable Petri nets is a particular net rewriting system where τ is a bijection. This model (even if formally equivalent to Petri nets) allows us to more precisely express systems in which structural dynamic changes can occur. However, automatic translation into Petri nets ensures that all the fundamental properties of Petri nets are still decidable for reconfigurable Petri nets. This model is thus amenable to automatic verification tools. However, the expansion into equivalent Petri nets may significantly increase the size of the net. Therefore, it may be more efficient to directly implement the methods of verification of properties of Petri nets on the original model. This presumes that we can define the notions of covering graphs, linear invariants, siphons, and traps directly for a reconfigurable Petri net, taking advantage of the symmetries induced by the group generated by transfer relations (which are bijections). We can also modify the definition of reconfigurable Petri nets, taking into account the net marking in rewriting rules, i.e., the net will only be reconfigured when a certain marking is reached. These topics are the subject of our current research.

In contrast, the entire class of net rewriting systems is Turing powerful and, thus, automatic verification is no longer possible in that case. However, this model is still interesting as a modeling and simulation tool. For some of the models described above, software tools have been developed for editing and simulating systems using a graphical user interface that permits a visualization of the current configuration of the system. These tools provide software support for the design of prototypes for dynamic concurrent systems. We are currently implementing a simulator for net rewriting systems.

Acknowledgments

The authors would like to thank Eric Badouel and the anonymous referees for their constructive comments and suggestions. This work has been supported by CICYT TIC 2001-2705-C03-01, by Acción Integrada Hispano-Alemana HA2001-0059 and by Proyecto UPV 7176.

References

[1] A. Asperti and N. Busi, "Mobile Petri Nets," Technical Report UBLCS-96-10, Univ. of Bologna, Italy, 1996.

[2] E. Badouel, P. Darondeau, and A. Tokmakoff, "Modelling Dynamic Agents Systems with Cooperating Automata," *Proc. Parallel and Distributed Processing Techniques and Applications (PDPTA '99),* pp. 11-17, 1999.

[3] E. Badouel, M. Llorens, and J. Oliver, "Modelling Concurrent Systems: Reconfigurable Nets," *Proc. Parallel and Distributed Processing Techniques and Applications (PDPTA'03),* vol. IV, pp. 1568-1574, 2003.

[4] E. Badouel and J. Oliver, "Reconfigurable Nets, a Class of High Level Petri Nets Supporting Dynamic Changes," Technical Report, Inria Research Report PI-1163, France, 1998.

[5] E. Badouel and J. Oliver, "Reconfigurable Nets, a Class of High Level Petri Nets Supporting Dynamic Changes within Workflow Systems," *Proc. Workflow Management: Net-Based Concepts, Models, Techniques and Tools (WFM '98),* CSR 98/07, Dept. of Math. and Computing Science, Eindhoven Univ. of Technology, pp. 129-145, 1998.

[6] E. Badouel and J. Oliver, "Dynamic Changes in Concurrent Systems: Modelling and Verification," Technical Report, Inria Research Report PI-3708, France, 1999.

[7] P. Baldan, "Modelling Concurrent Computations: From Contextual Petri Nets to Graph Grammars," PhD Thesis, Univ. of Pisa TD-1/00, 2000.

[8] M. Buscemi and V. Sassone, "High-Level Petri Nets as Type Theories in the Join Calculus," *Proc. Fourth Int'l Conf. Foundations of Software Science and Computation Structures (FoSSaCS '01),* pp. 104-120, 2001.

[9] A. Corradini, "Concurrent Computing: From Petri Nets to Graph Grammars," *Proc. Joint COMPUGRAPH/SEMAGRAPH Workshop Graph Rewriting and Computation,* 1995. http://www.elsevier. nl/locate/entcs/volume2.html.

[10] C. Dufourd, A. Finkel, and P. Schnoebelen, "Reset Nets between Decidability and Undecidability," *Proc. Int'l Colloquium Automata, Languages, and Programming (ICALP '98),* pp. 103-115, 1998.

[11] H. Ehrig, "Tutorial Introduction to the Algebraic Approach of Graph Grammmars," *Proc. Third Int'l Workshop Graph Grammars and Their Application to Computer Science,* pp. 3-14, 1987.

[12] J. Engelfriet, G. Leih, and G. Rozenberg, "Net Based Description of Parallel Object-Based Systems, or POTs and POPs," *Proc. Workshop Foundations of Object-Oriented Languages (FOOL '90),* pp. 229-273, 1991.
[13] C. Fournet and G. Gonthier, "The Reflexive Chemical Abstract Machine and the Join-Calculus," *Proc. 23rd ACM Symp. Principles of Programming Languages (POPL '96),* pp. 372-385, 1996.
[14] C. Fournet, G. Gonthier, J. Lévy, L. Maranget, and D. Rémy, "A Calculus of Mobile Agents," *Proc. Seventh Int'l Conf. Concurrency Theory (CONCUR '96),* pp. 406-421, 1996.
[15] P. Gradit, F. Vernadat, and P. Azéma, "Layered Δ-Net Specification of a Workshop," *Proc. Int'l Conf. Parallel and Distributed Processing Techniques and Applications (PDPTA '99),* vol. VI, pp. 2808-2814, 1999.
[16] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.
[17] M. Llorens and J. Oliver, "Modelización de Sistemas Concurrentes mediante Redes Reconfigurables," *IX Jornadas de Concurrencia,* pp. 213-224, 2001.
[18] R. Milner, J. Parrow, and D. Walker, "A Calculus of Mobile Processes," *J. Information and Computation,* vol. 100, no. 1, pp. 1-77, 1992.
[19] T. Murata, "Petri Nets: Properties, Analysis and Applications," *Proc. IEEE,* vol. 77, no. 4, pp. 541-580, 1989.
[20] J.L. Peterson, *Petri Net Theory and the Modeling of Systems.* Englewood Cliffs, N.J.: Prentice Hall, 1981.
[21] H. Schneider, "Graph Grammars as a Tool to Define the Behavior of Processes Systems: From Petri Nets to Linda," *Proc. Fourth Int'l Conf. Graph Grammars,* 1993.
[22] T.A. Sudkamp, *Languages and Machines. An Introduction to the Theory of Computer Science.* Addison-Wesley, 1988.
[23] R. Valk, "Self-Modifying Nets, a Natural Extension of Petri Nets," *Proc. Int'l Colloquium Automata, Languages, and Programming (ICALP '78),* pp. 464-476, 1978.
[24] R. Valk, "Generalizations of Petri Nets," *Proc. 10th Symp. Math. Foundations of Computer Science (MFCS '81),* pp. 140-155, 1981.
[25] F. Vernadat, K. Drira, P. Azéma, "An Integrated Description Technique for Distributed Cooperative Applications," *Proc. CESA'96 IMACS Multiconf., Symp. Discrete Events and Manufacturing Systems,* pp. 608-612, 1996.

Marisa Llorens received the PhD degree in computer science from the Universidad Politécnica de Valencia (Spain) in 2003. She is an associate professor for technical studies in the Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, Spain. Her current research interests include concurrent and distributed systems analysis, rewriting, and verification techniques.

Javier Oliver received the PhD degree in computer science from the Universidad Politécnica de Valencia, Spain, in 1996. He is an associate professor in the Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, Spain. His current research interests include concurrent and distributed systems analysis, rewriting, process algebras, and verification techniques.