From Time Petri Nets to Timed Automata: an Untimed Approach *

Davide D'Aprile[1], Susanna Donatelli[1], Arnaud Sangnier[2], and Jeremy Sproston[1]

[1]Dipartimento di Informatica, Università di Torino, 10149 Torino, Italy
[2]Lab. Spécification & Verification, ENS Cachan – CNRS UMR 8643, France
[1]{daprile,susi,sproston}@di.unito.it
[2]sangnier@lsv.ens-cachan.fr

Abstract. Time Petri Nets (TPN) and Timed Automata (TA) are widely-used formalisms for the modeling and analysis of timed systems. A recently-developed approach for the analysis of TPNs concerns their translation to TAs, at which point efficient analysis tools for TAs can then be applied. One feature of much of this previous work has been the use of timed reachability analysis on the TPN in order to construct the TA. In this paper we present a method for the translation from TPNs to TAs which bypasses the timed reachability analysis step. Instead, our method relies on the reachability graph of the underlying untimed Petri net. We show that our approach is competitive for the translation of a wide class of TPNs to TAs in comparison with previous approaches, both with regard to the time required to perform the translation, and with regard to the number of locations and clocks of the produced TA.

1 Introduction

As real-time systems become ever more complex and diffuse, it becomes increasingly important to develop methods for reasoning about such systems in a formal way. Two widely-used formalisms for the modeling and analysis of real-time systems are Time Petri Nets (TPNs) [15] and Timed Automata (TA) [3]. TPNs and TA are dense-time formalisms, which implies that their underlying state space is infinite, and therefore verification techniques which enumerate exhaustively the state space cannot be applied. In general, this difficulty is addressed by applying symbolic methods or by partitioning the infinite state-space. With regard to TA, the well-known region graph [3] or zone-based graph [2] techniques are two such methods, the latter of which forms the basis of the techniques implemented in tools such as UPPAAL [4, 18] and KRONOS [19, 12]. With regard to TPNs, in [5, 14] an approach based on the so-called state class graph (SCG) construction is presented. In the SCG the nodes are sets of states, represented by a pair comprising a marking and a firing domain, where the firing domain represents the set of times at which a transition can be fired. The SCG construction allows the verification of untimed reachability and LTL properties [5, 14], while variants of this method allow the verification of CTL, and a subset of TCTL [1] properties [6, 17].

A different approach to allow TCTL model checking of TPNs is to produce from a TPN a timed bisimilar TA which maintains TCTL properties, and then verify it by

* Supported in part by Miur project Firb-Perf and EEC project Crutial.

means of model-checking tools (for example, the above cited UPPAAL and KRONOS). In the literature there are two different techniques for the translation of TPNs to TA. The first is based on the Petri net (PN) structure [8], and is generally characterized by a potentially high number of clocks in the produced TA; the second is based on exploration of the timed state space, for example in [13], in which a method based on an extended version of the SCG is used to compute the so-called state class timed automaton (SCTA), and in [10], where zone-based timed reachability analysis (see [2]) allows the construction of the so-called marking timed automaton, that in the following we will call the zone-based marking timed automaton (ZBMTA). The ZBMTA always has no more locations and edges than the SCTA, while the latter has no more clocks than the former. Finally, it should be noticed that, in [10, 13], the reachability techniques for the generation of a TA are generally employed again subsequently to analyze the produced TA; this fact could increase the total verification time of the TPN under investigation.

In this paper we present a different technique for the translation of a TPN into a (strong) timed bisimilar TA, by using the reachability graph of the underlying *untimed* Petri Net to build what we have called the *marking class timed automaton* (MCTA). We will show that the SCTA, obtained by applying [13], and the MCTA, obtained by applying our approach, are incomparable in the number of locations and edges, while the MCTA produces a greater or equal number of locations and edges with respect to the ZBMTA approach, obtained by applying [10]; finally, the number of clocks may be equal to that of the SCTA, and less or equal to that of the ZBMTA. From these considerations it may be deduced that our approach represents a competitive choice for a number of classes of systems, especially when a trade-off is needed between the number of the produced locations and clocks; we will present experimental evidence to show this. The main disadvantage of our method is the requirement of boundedness of the underlying untimed PN, while [10, 13] require only TPN boundedness. In order to address this problem, we give some suggestions to partially bound specific PN subnets of the TPN under investigation. In addition, because our method may explore some paths in the untimed Petri net which are unreachable in the TPN, resulting in a greater number of locations, we consider an adjustment to the MCTA construction algorithm which, for some TPNs, can alleviate this problem.

This paper is organized as follows: Section 2 provides some background, while Section 3 explains our approach to verify TPNs by translation to TA, and makes a comparison with the SCTA and ZBMTA approaches. Section 4 presents some optimization techniques: a simple method to partially resolve the above cited unreachable path problem, a variant for reducing the number of locations of the produced TA, and some ideas to address the boundedness requirements of our approach. Section 5 describes our tool, GREATSPN2TA, for the translation of TPNs to TA in the input language of the KRONOS model checker, and reports some experimental results, obtained on a set of case studies, also comparing them against the results of the tool ROMEO [9, 16], which implements the SCTA and ZBMTA approaches. Section 6 concludes the paper.

2 Preliminaries

Timed Transition Systems. Let Σ be a finite set of *events*, and let $\mathbb{R}_{\geq 0}$ be the set of non-negative real numbers. A timed transition system (TTS) S is a tuple $\langle Q, q^0, \Sigma, \rightarrow \rangle$ where Q is the set of the *states*, $q^0 \in Q$ is the *initial state*, and $\rightarrow \subseteq Q \times (\Sigma \cup \mathbb{R}_{\geq 0}) \times Q$ is the set of *edges*. We use $q \xrightarrow{a} q'$ to denote $(q, a, q') \in \rightarrow$, which indicates that when the state of the system is q, it can change to q' upon *label* $a \in \Sigma \cup \mathbb{R}_{\geq 0}$. The edges labeled with an event of Σ are called *discrete edges* and the edges labeled with a non-negative real number are called *continuous edges*. A *path* is a finite or infinite sequence of edges $q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} \cdots$. A set of states $Q' \subseteq Q$ is *reachable* from a state q if there exists a finite path $q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{n-1}} q_n$, such that $q_0 = q$ and $q_n \in Q'$.

Timed Automata. Timed Automata (TA) [3] are automata extended with clocks, which are real-valued variables, and which increase at the same rate as real-time. Let X be a set of *clocks*, and $\Phi(X)$ be the set of the *clock constraints* over X, which are defined by the following grammar: $\varphi := x \leq c | x \geq c | x < c | x > c | \varphi_1 \wedge \varphi_2$, where $x \in X$ and $c \in \mathbb{Q}_{\geq 0}$ is a non-negative rational number. A *timed automaton* A is a tuple $\langle L, l^0, \Sigma, X, I, E \rangle$ where L is a finite set of *locations*, $l^0 \in L$ is an *initial location*, I is a (total) function $L \mapsto \Phi(X)$ that associates to each location an *invariant condition* (i.e. a clock constraint), and $E \subseteq L \times \Sigma \times \Phi(X) \times 2^X \times 2^{X^2} \times L$ represents the set of the *switches*. The switch $(l, \sigma, \varphi, \lambda, \rho, l') \in E$ represents a switch from l to l' on the event σ, with the guard φ (a clock constraint) describing the set of clock values that can enable the switch, the set $\lambda \subseteq X$ describing the clocks that are set to 0 by the switch, and $\rho \subseteq X^2$ describing how clocks should be renamed when the switch is taken. The semantics of TA is defined by means of a TTS, and its definition is standard (in particular, see [7, 13] for the semantics of the variant of TA with clock renaming); we omit it for reasons of space. In [3], the problems and the possible solutions regarding the infinite number of states and transitions of such a TTS are also illustrated. This leads to the use of abstraction methods, for example the *region graph* and the *zone graph*.

Time Petri Nets. A *Time Petri Net* (TPN) $\mathcal{T}$ [5, 15] is a tuple $\langle P, T, W^-, W^+, M_0, (\alpha, \beta) \rangle$ where $P = \{p_1, ..., p_m\}$ is a finite set of *places*, $T = \{t_1, ..., t_n\}$ is a finite set of *transitions*, $W^- : (\mathbb{N}^P)^T$ is the *backward incidence function*, $W^+ : (\mathbb{N}^P)^T$ is the *forward incidence function*, $M_0 \in \mathbb{N}^P$ is the *initial marking*, and $\alpha \in (\mathbb{Q}_{\geq 0})^T$ and $\beta \in (\mathbb{Q}_{\geq 0} \cup \{\infty\})^T$ are the *earliest* and *latest firing time* functions.

The semantics of a TPN $\mathcal{T}$ can be represented by a TTS $S_{\mathcal{T}}$. Before introducing the semantics we define the following notation. A *marking* M is an element of $\mathbb{N}^P$. In the following, we use standard notation for markings, such as $M \geq M'$ if and only if $M(p) \geq M'(p)$ for all $p \in P$, and $M - M'$ where $M - M'(p) = M(p) - M'(p)$ for all $p \in P$. A *valuation* is a vector $v \in (\mathbb{R}_{\geq 0})^n$ such that each value v_i represents the elapsed time since the last time transition t_i was enabled, or since the launching of the system if t_i was never enabled. The initial valuation $\bar{0} \in (\mathbb{R}_{\geq 0})^n$ is defined by $\bar{0}_i = 0$, for all $1 \leq i \leq n$. A transition t is said to be *enabled for a marking* M if and only if $M \geq W^-(t)$. For all $(t_k, M, t_i) \in T \times \mathbb{N}^P \times T$, let $\uparrow\!enabled(t_k, M, t_i) = (M - W^-(t_i) + W^+(t_i) \geq W^-(t_k)) \wedge ((M - W^-(t_i) < W^-(t_k)) \vee (t_k = t_i))$. Intuitively, $\uparrow\!enabled(t_k, M, t_i) = \texttt{true}$ if and only if t_k is newly enabled after the firing of t_i in M, where a transition t_k is said to be *newly enabled* after the firing of a transition t_i in M if t_k is not enabled for the marking $M - W^-(t)$ (or if $t_i = t_k$) and it is enabled for the marking $M' = M - W^-(t) + W^+(t)$.

The TTS $S_{\mathcal{T}} = \langle Q, q_0, T, \rightarrow \rangle$ associated to a TPN $\mathcal{T} = \langle P, T, W^-, W^+, M_0, (\alpha, \beta) \rangle$ is defined by $Q = \mathbb{N}^P \times (\mathbb{R}_{\geq 0})^n$, $q_0 = (M_0, \bar{0})$, and $\rightarrow \in Q \times (T \cup \mathbb{R}_{\geq 0}) \times Q$ is the set of edges defined by:

1. The *discrete edges* are defined by, for all $t_i \in T$:

$$(M, v) \xrightarrow{t_i} (M', v') \Leftrightarrow \begin{cases} M \geq W^-(t_i) \wedge M' = M - W^-(t_i) + W^+(t_i) \\ \alpha(t_i) \leq v_i \leq \beta(t_i) \\ v'_k = \begin{cases} 0 & \text{if } \uparrow enabled(t_k, M, t_i) \\ v_k & \text{otherwise.} \end{cases} \end{cases}$$

2. The *continuous edges* are defined by, for all $\delta \in \mathbb{R}_{\geq 0}$:

$$(M, v) \xrightarrow{\delta} (M, v') \Leftrightarrow v' = v + \delta, \text{ and } \forall k \in \{1, \cdots, n\}, (M \geq W^-(t_k) \Rightarrow v'_k \leq \beta(t_k)) .$$

The last condition on continuous transitions ensures that the time that elapses in a marking cannot increase to a value which would disable transitions that are enabled by the marking. For TPNs, as for TA, it is not possible to work directly on the TTS which represents the behavior of the TPN, because this TTS has infinitely many states (and infinitely many labels). Again, the use of abstraction methods permit the construction of a transition system where the labels expressing the passing of time are eliminated and where states are regrouped into classes on which the reachability analysis can be done. The *state class graph* [5] and the *zone graph* [10] are examples of such an approach. However these methods do not always give a result because, as pointed out in [5], for a TPN the problems of reachability and of boundedness are undecidable.

3 From Time Petri Nets to Timed Automata

We now describe our approach for translating a TPN model into a TA, called the marking class timed automaton (MCTA), in order to subsequently perform analysis on MCTA. Section 3.1 is devoted to this technique, also providing a proof that the TTS of the TPN and of the MCTA are timed bisimilar, while in Section 3.2 our approach is compared with those based on the SCTA and the ZBMTA [13, 10].

3.1 MCTA of a TPN

In this section we present the MCTA construction, where the constructed TA has an equivalent (timed bisimilar) behavior to that of a TPN. Consider the TPN $\mathcal{T} = \langle P, T, W^-, W^+, M_0, (\alpha, \beta) \rangle$. We will "untime" the TPN $\mathcal{T}$ (that is, remove the timing functions (α, β)) in order to obtain a Place/Transition PN $\mathcal{P} = \langle P, T, W^-, W^+, M_0 \rangle$. We denote by $\mathcal{R}_u(M_0) \subseteq \mathbb{N}^P$ the reachability set of $\mathcal{P}$ (the set of markings that $\mathcal{P}$ can reach from its initial marking M_0). When bounded (i.e. $(\exists k \in \mathbb{N})(\forall p \in P)(\forall M \in \mathcal{R}_u(M_0))(M(p) \leq k)$), the behavior of this PN can be represented by the *reachability graph* (RG), which is an untimed finite-state transition system $\langle Q, q_0, T, \rightarrow \rangle$ where

$Q = \mathcal{R}_u(M_0)$, $q_0 = M_0$, and the edge relation $\rightarrow$ is defined by classical 1-step reachability in untimed PN: for all $M, M' \in \mathcal{R}_u(M_0)$, for all $t \in T$:

$$M \xrightarrow{t} M' \Leftrightarrow M \geq W^-(t) \text{ and } M' = M + W^+(t) - W^-(t) \ .$$

The MCG **construction.** We now present the algorithm which builds the *marking class graph* (MCG) $\Gamma(\mathcal{T})$ of the TPN $\mathcal{T}$, which is a transition system $\Gamma(\mathcal{T}) = \langle \mathcal{C}, C_0, T, \rightarrow_{mc} \rangle$. The states $\mathcal{C}$ of $\Gamma(\mathcal{T})$ are called *marking classes*. Each marking class is a triple of the form $\langle M, \chi, trans \rangle$, comprising a marking M of $\mathcal{T}$, a set χ of clocks, and a function $trans : \chi \rightarrow 2^T$ associating a set of transitions to each clock in χ. The initial marking class $C_0 = \langle M_0, \{x_0\}, trans_0 \rangle$ is such that M_0 is the initial marking of $\mathcal{T}$, the set of clocks of C_0 is composed of a single clock x_0, and $trans_0$ is defined by $trans_0(x_0) = \{t \in T \mid t \text{ is enabled for } M_0\}$. To build the graph, we also need the notion of clock similarity (adapted from [13]), in order to group certain marking classes together. Two marking classes $C = \langle M, \chi, trans \rangle$ and $C' = \langle M', \chi', trans' \rangle$ are *clock similar*, denoted $C \approx C'$, if and only if they have the same markings, the same number of clocks and their clocks are mapped to the same transitions, written formally as:

$$C \approx C' \Leftrightarrow M = M', |\chi| = |\chi'| \text{ and } \forall x \in \chi, \exists x' \in \chi', trans(x) = trans(x') \ .$$

The MCG construction is shown in Algorithm 1, and is a classical breadth-first graph generation algorithm which starts from the initial unexplored marking class C_0. At each step an unexplored marking class C is marked as explored, all marking classes C' reachable in one step (firing of a transition) from C are added to the set of unexplored classes, unless an equivalent one (according to clock similarity) has already been considered before. The algorithm terminates when all unexplored markings have been considered. In lines 1.6 to 1.13, the set of clocks χ' and the function $trans'$, which associates clocks to enabled transitions, are computed. We note that the construction of this graph can be done by following the different paths in the reachability graph of the underlying PN adding a clock set χ' and a relation $trans'$, and possibly "unlooping" some loops of the reachability graph when a marking is reached many times with associated marking classes which are not clock-similar.

The MCTA **Construction.** From the MCG defined above, it is possible to build a TA $A(\mathcal{T})$ which has the same behavior as the TPN $\mathcal{T}$, as we will show in the next section. Let $\mathcal{T} = \langle P, T, W^-, W^+, M_0, (\alpha, \beta) \rangle$ be a TPN and $\Gamma(\mathcal{T}) = \langle \mathcal{C}, C_0, T, \rightarrow_{mc} \rangle$ its associated marking class graph. The *marking class timed automaton* (MCTA) $A(\mathcal{T})$ associated to $\mathcal{T}$ is the TA $\langle L, l_0, \Sigma, X, I, E \rangle$ defined by:

- $L = \mathcal{C}$ is the set of the marking classes;
- $l_0 = C_0$, where C_0 is the initial marking class ($C_0 = \langle M_0, \{x_0\}, trans_0 \rangle$);
- $X = \bigcup_{\langle M, \chi, trans \rangle \in \mathcal{C}} \chi$;
- $\Sigma = T$;
- E is the set of switches defined by:

$$\forall C_i = \langle M_i, \chi_i, trans_i \rangle \in \mathcal{C}, \forall C_j = \langle M_j, \chi_j, trans_j \rangle \in \mathcal{C}$$
$$\exists C_i \xrightarrow{t_i}_{mc} C_j \Leftrightarrow \exists (l_i, a, \phi, \lambda, \rho, l_j) \in E \text{ such that}$$
$$\begin{cases} l_i = C_i, l_j = C_j, a = t_i, \\ \phi = (trans_i^{-1}(t_i,) \geq \alpha(t_i)),\ \lambda = \{trans_j^{-1}(t_k) \mid \uparrow enabled(t_k, M_i, t_i) = \mathtt{true}\}, \\ \forall x \in \chi_i, \forall x' \in \chi_j,\ \text{such that } trans_j(x') \subseteq trans_i(x), x' \notin \lambda, \rho(x') = x\ ; \end{cases}$$

input : The initial marking class C_0 of a TPN $\mathcal{T}$
output: The MCG of $\mathcal{T}$

$MCG := \emptyset$; $New := C_0$;
while *New is not empty* **do**
  $C :=$ `remove`(New); (where $C = \langle M, \chi, trans \rangle$)
  $Fireable(C) := \{t \mid t$ is enabled for $M\}$;
  for *all transitions* $t \in Fireable(C)$ **do**
    $M' := M + W^+(t) - W^-(t)$;
    For each clock $x \in \chi$, remove from $trans(x)$ all the transitions t_k such that t_k is enabled in M and is not in $M - W^-(t)$, to obtain a relation $trans'$;
    The clocks whose image by $trans'$ is empty are removed from χ, to obtain a set of clocks χ';
    for *all transitions* t_k *which verify* $\uparrow enabled(t_k, M, t) =$ `true` **do**
      if *a clock x has already been created for the computation of* C' **then**
        t_k is added to $trans'(x)$;
      else
        a new clock x_n is created; n is the smallest available index among the clocks of χ' and $trans'(x_n) = t_k$;
      end
    end
    $C' := \langle M', \chi', trans' \rangle$;
    if *there is a marking class* C'' *in MCG such that* $C' \approx C''$ **then**
      $MCG := MCG \cup \{C \xrightarrow{t}_{mc} C''\}$;
    else
      $MCG := MCG \cup \{C \xrightarrow{t}_{mc} C'\}$ and `add`(New, C');
    end
  end
end

Algorithm 1: MCG construction

– $\forall C_i = \langle M_i, \chi_i, trans_i \rangle \in \mathcal{C}$, $I(C_i) = \bigwedge_{x \in \chi_i, t \in trans_i(x)} (x \leq \beta(t))$.

In order to build the MCTA of a TPN, the number of marking classes has to be bounded, otherwise the construction of the MCG will not terminate. Note that the MCG has a bounded number of marking classes if and only if the underlying untimed PN is bounded. We recall that in contrast to the case of the boundedness of TPN [5], the boundedness of a PN is decidable. We will return to boundedness issues in Section 4.3.

Bisimulation. We now define an equivalence relation between the states of the TPN $\mathcal{T}$ and the states of its associated MCTA, and we will prove that this relation is a timed bisimulation. Our results are analogous to those in the context of the SCTA [13] and the ZBMTA [10].

First, we recall the definition of timed bisimulation (see, for example, [8, 13, 10]). Let $S_1 = \langle Q_1, q_1^0, \Sigma_1, \rightarrow_1 \rangle$ and $S_2 = \langle Q_2, q_2^0, \Sigma_2, \rightarrow_2 \rangle$ be two TTSs. The equivalence relation $\approx \subseteq Q_1 \times Q_2$ on Q_1 and Q_2 is a *timed bisimulation* if and only if, for all $a \in \Sigma \cup \mathbb{R}_{\geq 0}$:

– if $s_1 \approx s_2$ and $s_1 \xrightarrow{a} s_1'$ then there exists $s_2 \xrightarrow{a} s_2'$ such that $s_1' \approx s_2'$;

– if $s_1 \approx s_2$ and $s_2 \xrightarrow{a} s'_2$ then there exists $s_1 \xrightarrow{a} s'_1$ such that $s'_1 \approx s'_2$.

Let $\mathcal{T} = \langle P, T, W^-, W^+, M_0, (\alpha, \beta)\rangle$ be a TPN and $A(\mathcal{T})$ its associated MCTA. We consider $\overline{Q}_{\mathcal{T}}$ the set of reachable states of $\mathcal{T}$ and Q_A the set of states of $A(\mathcal{T})$. We define the relation $\simeq_{mc} \subseteq \overline{Q}_{\mathcal{T}} \times Q_A$ by the following rule. For all $s = (M, \nu_{\mathcal{T}}) \in \overline{Q}_{\mathcal{T}}$, for all $r = (C_r, v_A) \in Q_A$ (with $C_r = \langle M_r, \chi_r, trans_r\rangle$):

$$s \simeq_{mc} r \Leftrightarrow \begin{cases} M = M_r \text{ and} \\ \forall t \in T \text{ such that } t \text{ is enabled in } M, \\ \qquad \nu_{\mathcal{T}}(t) = v_A(x) \text{ with } x \in \chi_r \text{ such that } t \in trans_r(x) \ . \end{cases}$$

Theorem 1. *The binary relation $\simeq_{mc} \subset \overline{Q}_{\mathcal{T}} \times Q_A$ is a timed bisimulation.*

If we consider a TPN $\mathcal{T} = \langle P, T, W^-, W^+, M_0, (\alpha, \beta)\rangle$ and its associated MCTA $A(\mathcal{T})$, because we have by construction $(M_0, \overline{0}) \simeq_{mc} (C_0, \overline{0})$, we conclude that a marking M is reachable from M_0 in $\mathcal{T}$ if and only if there exists a state of $A(\mathcal{T})$ whose associated marking (within the state's marking class) is M. The timed bisimulation property also allows us to obtain the set of states of $\mathcal{T}$ which satisfy a TCTL property: the TCTL property can be verified on $A(\mathcal{T})$, and the resulting set of states of $\mathcal{T}$ satisfying the property can be obtained using $\simeq_{mc}$.

Example. We now consider the application of our procedure to the TPN of Figure 1. The corresponding MCTA is given in Figure 2. The structure (locations, represented by nodes, and switches, represented by arcs) of the MCTA is derived from the MCG, which provides also the following information:

- for every location, information regarding the corresponding marking of the considered (PN underlying the) TPN, as well as information about which clock is linked to the currently enabled transitions in the corresponding state of the original model;
- for every arc, the transition which fires in the TPN.

The MCTA construction step labels the locations with invariants, while guards, clock resets and clock renaming functions are added to the arcs. Guards are written above the line labeling each arc, whereas resets and clock renaming are indicated below. Starting from the initial location C_0, we have two newly enabled transitions, t_1 and t_2, to which an unique clock, named x, is assigned; the corresponding invariants and guards, indicated on the corresponding outgoing arcs, are defined with respect to the timing intervals in the TPN under translation. When the outgoing arc labeled t_1 is taken from location C_0 to location C_1 (between time 4 and 5), the transition named t_2 is still enabled, so the clock x remains assigned to t_2, and must not be reset before entering C_1. In location C_1 the automaton cycles forever, taking the arc labeled t_2 every 1 time unit, and always resetting the clock x before entering the same location, because t_2 is always newly enabled after each firing. When the outgoing arc labeled t_2 is taken from location C_0 to location C_2 (after exactly 1 time unit), the transition named t_1 is still enabled, so the clock x remains assigned to it (and x is not reset), while the fired transition t_2 is newly enabled, and so is assigned to a new clock, y, which must be reset before entering C_2. In location C_2 the automaton can cycle every 1 time unit, resetting the clock y on every cycle, because t_2 is always newly enabled after each firing. When the outgoing

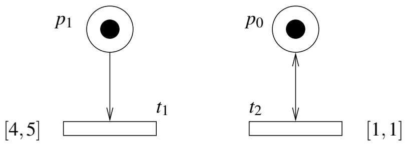
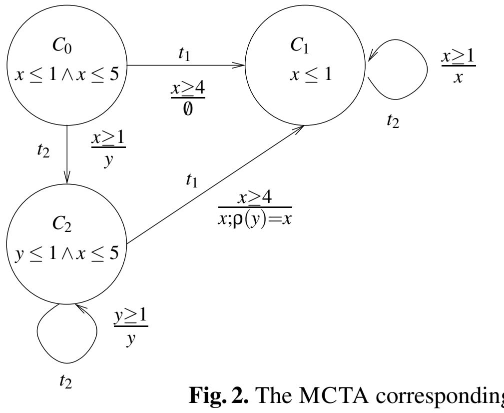


Fig. 1. A TPN model $\mathcal{T}$



$\mathcal{C}$	M	$trans$
C_0	p_0+p_1	$x \leftarrow \langle t_1, t_2 \rangle$
C_1	p_0	$x \leftarrow \langle t_2 \rangle$
C_2	p_0+p_1	$x \leftarrow \langle t_1 \rangle$ $y \leftarrow \langle t_2 \rangle$

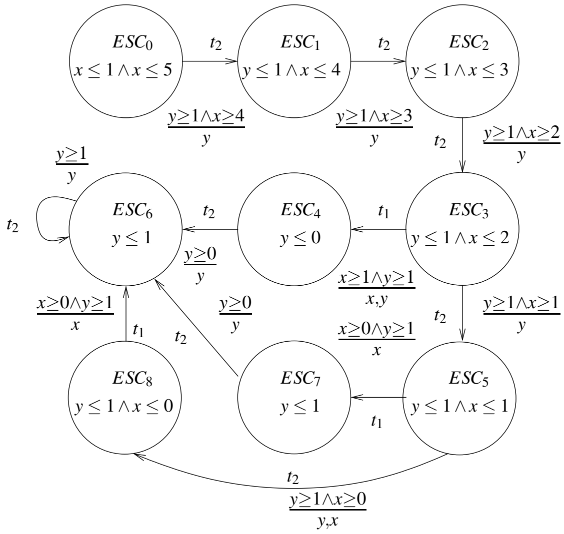
Fig. 2. The MCTA corresponding to TPN $\mathcal{T}$ in Figure 1

arc labeled t_1 is taken from location C_2 to location C_1 (after between 4 and 5 time units since t_1 was enabled), the transition named t_2 is still enabled, but in C_1 the transition t_2 is already assigned to a clock named x; this implies that the clock y must be renamed to x while taking the arc. Note that the guard on the arc between C_0 and C_1 is never true, due to the invariant associated with C_0, but that C_1 is reachable via C_2.

3.2 Comparing the MCG, ESCG, and ZBMCG approaches

In this section we compare the ESCG, MCG, and ZBMCG approaches, taking into account the cardinality of locations and edges, as well as the number of clocks of the produced TA. We recall that, with respect to the MCG, the ESCG nodes are enriched by the firing domain constraints [13], while in the ZBMCG nodes the available information regards only the reached markings [10].

We first observe that *the* MCG *and the* ESCG *approaches are incomparable with respect to the number of generated locations*. We provide two examples to substantiate this remark. Let $|\text{MCG}|_{\mathcal{T}}$ and $|\text{ESCG}|_{\mathcal{T}}$ be the cardinality of locations of the MCG and ESCG, respectively, of the TPN $\mathcal{T}$ of Figure 1. The fact that $|\text{MCG}|_{\mathcal{T}} = 3$ can be derived from the TA shown in Figure 2, while the ESCG construction for $\mathcal{T}$ leads to $|\text{ESCG}|_{\mathcal{T}} = 9$. The TA corresponding to the ESCG is shown in Figure 3. The table of Figure 3 defines, for each extended class ESC, the net marking M, the association *trans* of transitions to clocks, and the firing domains D of transitions. It is clear that, in this net, the ESCG construction distinguishes more than the MCG one. This happens because, in the ESCG, for each reachable marking there may be a number of associated firing domains. Figure 4, instead, give us an example of a TPN $\mathcal{T}$ for which $|\text{MCG}|_{\mathcal{T}} \geq$



ESC	M	D	$trans$
ESC_0	p_0+p_1	$4 \le t_1 \le 5$ $1 \le t_2 \le 1$ $3 \le t_1 - t_2 \le 4$	$x \leftarrow \langle t_1, t_2 \rangle$
ESC_1	p_0+p_1	$3 \le t_1 \le 4$ $1 \le t_2 \le 1$ $2 \le t_1 - t_2 \le 3$	$x \leftarrow \langle t_1 \rangle$ $y \leftarrow \langle t_2 \rangle$
ESC_2	p_0+p_1	$2 \le t_1 \le 3$ $1 \le t_2 \le 1$ $1 \le t_1 - t_2 \le 2$	$x \leftarrow \langle t_1 \rangle$ $y \leftarrow \langle t_2 \rangle$
ESC_3	p_0+p_1	$1 \le t_1 \le 2$ $1 \le t_2 \le 1$ $0 \le t_1 - t_2 \le 1$	$x \leftarrow \langle t_1 \rangle$ $y \leftarrow \langle t_2 \rangle$
ESC_4	p_0	$0 \le t_2 \le 0$ $-1 \le t_1 - t_2 \le -1$	$y \leftarrow \langle t_2 \rangle$
ESC_5	p_0+p_1	$0 \le t_1 \le 1$ $1 \le t_2 \le 1$ $-1 \le t_1 - t_2 \le 0$	$x \leftarrow \langle t_1 \rangle$ $y \leftarrow \langle t_2 \rangle$
ESC_6	p_0	$1 \le t_2 \le 1$	$y \leftarrow \langle t_2 \rangle$
ESC_7	p_0	$0 \le t_2 \le 1$	$y \leftarrow \langle t_2 \rangle$
ESC_8	p_0+p_1	$0 \le t_1 \le 0$ $1 \le t_2 \le 1$ $t_1 - t_2 = -1$	$x \leftarrow \langle t_1 \rangle$ $y \leftarrow \langle t_2 \rangle$

Fig. 3. The SCTA corresponding to TPN $\mathcal{T}$ in Figure 1

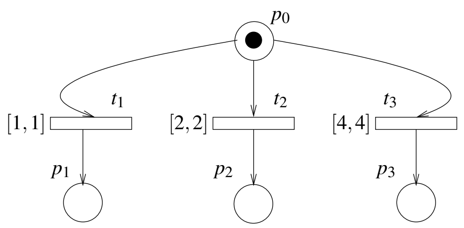
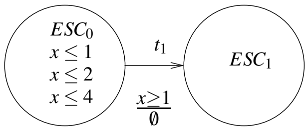


Fig. 4. A TPN model $\mathcal{T}$, with $|\mathrm{MCG}|_{\mathcal{T}} \ge |\mathrm{ESCG}|_{\mathcal{T}}$

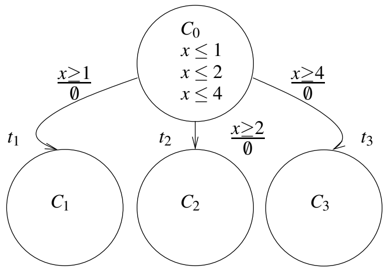
$|\mathrm{ESCG}|_{\mathcal{T}}$, as shown in Figures 5 and 6. In this case, the MCG algorithm, being unable to identify unreachable paths, produces an higher number of locations, two of which are unreachable in the MCTA. In fact, the ESCG construction process, thanks to the firing domain computation, correctly "cuts off" the untakeable t_2 and t_3 transitions, and so the C_2 and C_3 locations are not reached, while this does not happen with the MCG.

Next, we observe that *the* ZBMCG *approach results in no more locations and switches than the* MCG *and* ESCG *approaches*. The ZBMCG method generates only those markings that are reachable in the TPN, whereas our MCG approach generates markings that are reachable in the underlying untimed PN. For this reason alone, it is easy to show an example in which the number of locations and switches produced by the ZBMCG method is less than or equal to the number of locations and switches produced by our MCG method. Now note that each location produced by the ZBMCG method corresponds to a set of locations produced by the ESCG method: the markings corresponding to the locations will be the same, but, in the case of the ESCG method, the locations are enriched with firing domains. A similar argument can be used for the



ESC	M	D	$trans$
ESC_0	p_0	$t_1 = 1$ $t_2 = 2$ $t_3 = 4$ $t_2 - t_1 = 1$ $t_3 - t_2 = 2$ $t_3 - t_1 = 3$	$x \leftarrow \langle t_1, t_2, t_3 \rangle$
ESC_1	p_1	$\emptyset$	$\emptyset$

Fig. 5. The SCTA corresponding to TPN $\mathcal{T}$ in Figure 4



C	M	$trans$
C_0	p_0	$x \leftarrow \langle t_1, t_2, t_3 \rangle$
C_1	p_1	$\emptyset$
C_2	p_2	$\emptyset$
C_3	p_3	$\emptyset$

Fig. 6. The MCTA corresponding to TPN $\mathcal{T}$ in Figure 4

switches. Taking again the TPN as in Figure 1, in Figure 7 we give the TA obtained by applying the ZBMCG technique.

Despite the fact that the ESCG and the ZBMCG can result in smaller TA in terms of locations and switches or clocks than the MCG, we show in Section 5 that, when applied to a number of examples from the literature, the proposed MCG-based translation can be competitive in size and execution time.

4 Improving the effectiveness of the MCG approach

In this section we present some modifications of the MCG algorithm, in order to improve the effectiveness and applicability of our proposed solution.

4.1 Reducing the number of unreachable locations

The first modification allows to "cut off" paths that could obviously not be taken, such as the firing of t_1 in C_0 of the example in Figures 1 and 2. As observed before in the TPN of Figure 1, when t_1 and t_2 are newly enabled only t_2 can fire. Cutting off the edge from C_0 to C_1 does not change $|\mathrm{MCG}|_{\mathcal{T}}$ in this case, but it does for the TPN of Figure 4, because it discards the possibility of firing t_2 and t_3. Line 1.4 of the algorithm can be changed to check the earliest and latest firing time of the newly enabled transitions, and to remove from consideration transitions that are not firable:

$$\begin{aligned} Fireable(C) := &\ \{t \mid t \text{ is enabled for } M_C\} \setminus \\ &\ \{t \mid \exists t' \in T. \exists x \in \chi_C \text{ such that } t, t' \in trans_C(x) \text{ and } \alpha(t) > \beta(t')\}; \end{aligned}$$

Observe that this modification takes timing information into account, as ESCGs and ZBMCGs do, but with the difference that the check does not consider the elapsed enabling time (which is encoded in the state class domains in ESCGs, and in zones in

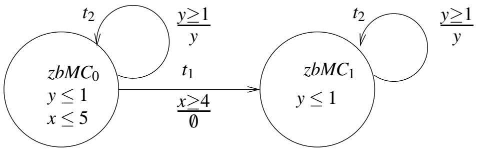
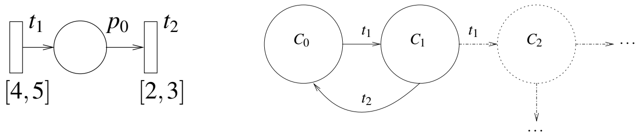


Fig. 7. The ZBMTA corresponding to TPN $\mathcal{T}$ in Figure 1



C	M	*trans*
C_0	$\emptyset$	$x \leftarrow \langle t_1 \rangle$
C_1	p_0	$x \leftarrow \langle t_1, t_2 \rangle$
C_2	$2p_0$	$x \leftarrow \langle t_2 \rangle$ $y \leftarrow \langle t_1 \rangle$

Fig. 8. A TPN model $\mathcal{T}$, for which the application of the local optimization is useful

ZBMCGs). The TPN on the left part of Figure 8 illustrates an effective case of the modification of the algorithm: the original MCG is infinite (since $\mathcal{R}_u$ is unbounded), but the modified algorithm stops because, as shown on the central and right part of Figure 8, the firing of t_1 in C_1 is not considered.

4.2 Trading clocks for locations and speed

Our second modification increases the number of clocks, but decreases the number of locations and the computation time. This variant to the MCTA generation procedure consists of the assignment of a unique clock for every enabled transition, and not a unique clock for every set of newly enabled transitions: indeed, unless two transitions are always enabled at the same time, it is better to associate to them two separate clocks. As a consequence the expensive check of clock similarity can be removed from the algorithm. We call MCTA^{clock} the automata obtained with such a procedure. The construction of the MCTA^{clock} of the TPN of Figure 1 results in the same TA as that corresponding to the ZBMTA and is shown in Figure 7: assigning two different clocks, x and y, to the newly enabled transitions t_1 and t_2 in location C_0 let us merge C_0 and C_2 into a unique location, decreasing from 3 to 2 the number of required locations.

4.3 Dealing with unboundedness

Consider the TPN on the left part of Figure 9, illustrating a producers-consumers system model distributed with the ROMEO package. The set $\mathcal{R}_u$ of this net is unbounded, but the TPN itself has a bounded behavior because the consumers (top part of the net) are always faster than the producers, so that tokens never accumulate unboundedly in place $P3$. Observe that in TPN models whose boundedness depends of time, even the smallest change in the definition of the timing constraints may cause non-termination of the ESCG and ZBMCG algorithms; on the other hand such models may be of interest in many application fields. The method we propose here, inspired by similar techniques for performance evaluation of unbounded stochastic Petri Nets, is to artificially bound

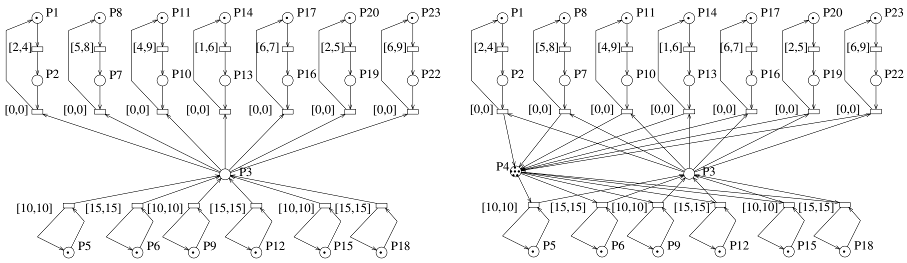


Fig. 9. An unbounded TPN (left), and the same model after the bounding procedure (right)

the net, using an initial, random guess for this bound, and then to check on the corresponding TA whether the bound is too low. We proceed as follows:

1. Compute the P-semiflows of the untimed PN.
2. If all places are covered by at least one P-semiflow, then the net is bounded and we can apply the MCG algorithm in the standard way; otherwise, for all places p_i not covered by a P-semiflow, build the complementary places $\bar{p}_i$, and set $M_0(\bar{p}_i)$ to a guessed value (we use $\bar{P}$ to denote the set of complementary places).
3. Build the MCTA using Algorithm 1.
4. Finally, check on the MCTA whether there is a reachable state of the TA of marking M, in which the complementary place is actually limiting the original timed behavior (formally, $\exists t$: $\forall p \in P, M(p) \geq W^-(p,t) \wedge \exists p \in \bar{P}, M(p) < W^-(p,t)$); if such a state exists, increase the initial guess for $M_0(\bar{p})$ and repeat.

Note that, if the TPN is unbounded, then the number of iterations is unbounded and the algorithm does not terminate (the ESCG and ZBMCG computations also do not terminate). P-semiflow and complementary places are standard PN concepts, and we do not recall them here. We only show how the net on the left part of Figure 9 is modified to obtain the net on the right part of the same figure. P-semiflow analysis reveals that place $P3$ is unbounded and the complementary place $P4$ is inserted. Choosing $M_0(P4) = 6$ bounds also $P3$ to a maximum of 6 tokens. The check on the MCTA reveals that this was a good choice, and we can safely use the MCTA built from the net on the right part of Figure 9, rather than the TPN on the left part of the figure (the underlying PN of which is unbounded), because they have the same behavior over reachable states.

5 The GREATSPN2TA tool

In this section we present the tool GREATSPN2TA for the computation of the MCTA (or MCTA^{clock}) of a given TPN. The underlying PN is described with the tool GREATSPN [11], which is a software package for the modeling, validation and performance evaluation of distributed systems using models based on stochastic Petri nets. The produced MCTA (or MCTA^{clock}) is described in the input format of KRONOS [12], a model-checking tool for TA. In the following, we compare GREATSPN2TA to

ROMEO. The software ROMEO [9] permits the state space computation of TPN, on-the-fly TCTL model-checking and the translation from TPN to TA with equivalent behavior. ROMEO incorporates two tools of interest in our context, namely GPN and MERCUTIO. Both tools transform a given TPN to the UPPAAL or KRONOS input format: the tool GPN exploits the SCTA computation, whereas MERCUTIO is based on the ZBMTA construction.

We ran MERCUTIO, GPN, and GREATSPN2TA (using also the variant GREATSPN2TA^{clock}, which implements the MCTA^{clock} construction), on a number of different models. Our experiments were executed on a 1.60 GHz Pentium 4 PC with 512 MB of RAM, running Linux. Table 1 lists, for every model, the number of locations and clocks of the TA, and the elapsed time to compute the TA. We considered two classical PN models: the dining philosophers (with 4 philosophers, *Philo4*), the slotted ring with 4 devices ($SR4$), and three models taken from the ROMEO package: a producer-consumer with 6 producers and 7 consumers ($P6C7$), and a set of parallel sequences ($Oex15$), which we have also modified so that each sequence cycles ($Oex15^{cycle}$). For *Philo4* and $Oex15^{cycle}$ a number of different timings of the TPN were considered: in the *Philo4* case, we have forced one of the four philosophers to be 10, 100, or 1000 times slower during the thinking activity (so obtaining the $Timing_{1slw-10}$, $Timing_{1slw-100}$, and $Timing_{1slw-1000}$ variants, respectively); in the $Oex15^{cycle}$ case, the time intervals describing the different activities were considered totally disjoint ($Timing_{disj}$), partially overlapping ($Timing_{overlapping}$), or having the same latest firing times ($Timing_{contained-LFT}$). The results, shown in Table 1, provide examples of the various trade-off that the four methods offer. Due to the different characteristics of the four algorithms, we compare the tools by pairs: GREATSPN2TA with GREATSPN2TA^{clock}, GPN with MERCUTIO, GPN with GREATSPN2TA and MERCUTIO with GREATSPN2TA^{clock}.

GREATSPN2TA and GREATSPN2TA^{clock}. GREATSPN2TA always produces a greater number of locations and a smaller number of clocks than the GREATSPN2TA^{clock} variant: the smaller number of clocks is nevertheless paid for in terms of execution time, especially for models in which, in each state, there is an high number of enabled transitions (indeed the execution of GREATSPN2TA on $P6C7$ did not terminate even after 5 minutes). The greater number of locations can be explained by recalling the discussion of Section 4.2. As expected, execution times do not change when changing the timing of transitions.

GPN and MERCUTIO. As already observed, GPN optimizes clocks and MERCUTIO optimizes locations: there is not a definitive winner in terms of execution times, although they are both sensitive to timings (most notably in the *Philo4* case).

GPN and GREATSPN2TA. For the examples considered, the two tools generate the same number of clocks. In the $P6C7$ case the MCTA computation explodes while computing clock similarity, due to the high number of transitions enabled in each state. In all other cases, the execution time is smaller for GREATSPN2TA.

MERCUTIO and GREATSPN2TA^{clock}. MERCUTIO assigns statically one clock per transition and leaves to the TA tool (UPPAAL or KRONOS) the task of minimizing the number of clocks, while GREATSPN2TA^{clock} assigns a different clock to each enabled transition: this explains the smaller number of clocks in the

Model	GPN	MERCUTIO	GREATSPN2TA	GREATSPN2TA^{clock}
*SR*4	22907 *loc*	5136 *loc*	7327 *loc*	5136 *loc*
	4 *clocks*	33 *clocks*	4 *clocks*	8 *clocks*
	4.30 *s*	3.86 *s*	2.63 *s*	2.08 *s*
*Philo*4	4406 *loc*	322 *loc*	1161 *loc*	322 *loc*
	6 *clocks*	17 *clocks*	6 *clocks*	8 *clocks*
	1.50 *s*	0.16 *s*	0.11 *s*	0.07 *s*
$Timing_{1slw-10}$	6.7 *s*	6.2 *s*	0.11 *s*	0.07 *s*
$Timing_{1slw-100}$	> 300 *s*	> 300 *s*	0.11 *s*	0.07 *s*
$Timing_{1slw-1000}$	> 300 *s*	> 300 *s*	0.11 *s*	0.07 *s*
P6C7	11490 *loc*	449 *loc*	*n.a.*	896 *loc*
	3 *clocks*	21 *clocks*	*n.a.*	13 *clocks*
	3.44 *s*	4.70 *s*	> 300 *s*	1.24 *s*
*Oex*15	1048 *loc*	360 *loc*	625 *loc*	625 *loc*
	4 *clocks*	17 *clocks*	4 *clocks*	4 *clocks*
	0.36 *s*	0.63 *s*	0.12 *s*	0.11 *s*
$Oex15^{cycle}$	3510 *loc*	256 *loc*	369 *loc*	256 *loc*
	4 *clocks*	17 *clocks*	4 *clocks*	4 *clocks*
	3.10 *s*	7.9 *s*	0.07 *s*	0.06 *s*
$Timing_{disjoint}$	7.8 *s*	32.5 *s*	0.07 *s*	0.06 *s*
$Timing_{overlapping}$	4.7 *s*	32.7 *s*	0.07 *s*	0.06 *s*
$Timing_{contained-LFT}$	4.8 *s*	25.9 *s*	0.07 *s*	0.06 *s*

Table 1. Experiments results for GPN, MERCUTIO, GREATSPN2TA, and GREATSPN2TA^{clock}.

GREATSPN2TA^{clock} column. As expected, the number of locations is smaller in MERCUTIO (which is optimal in this respect), but its execution times can be much worse, especially when changing transition timings.

6 Conclusions

In this paper we have presented a method to translate a TPN to a TA by exploiting reachability of the underlying untimed PN of the TPN. This technique has a disadvantage that the untimed PN can be unbounded, even if the TPN is bounded; to address this issue, we have described an empirical method for bounding the PN using complementary places, and then checking if this bound is too restrictive. The experimental results show that the computation time used by our method is competitive for a number of classes of system, and the produced TA generally offer a good compromise between the number of locations and the number of clocks. In future work, we plan to address methods for obtaining information about bounds on the number of tokens in places of the TPN, which can then be used in our approach based on complementary places. We also intend to implement a translation to UPPAAL TA (which requires a translation of the MCTA, which has clock renaming, to an equivalent TA without renaming [7]), and to consider the use of clock reduction, as implemented in model-checking tools for TA, in the context of our technique.

References

1. R. Alur, C. Courcoubetis, and D. L. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.

2. R. Alur and D. Dill. Automata-theoretic verification of real-time systems. *Formal Methods for Real-Time Computing*, pages 55–82, 1996.
3. R. Alur and D. L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.
4. G. Behrmann, A. David, K. G. Larsen, J. Håkansson, P. Pettersson, W. Yi, and M. Hendriks. UPPAAL 4.0. In *Proceedings of the 3rd International Conference on Quantitative Evaluation of Systems (QEST 2006)*, pages 125–126. IEEE Computer Society Press, 2006.
5. B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using time Petri nets. *IEEE Transactions on Software Engineering*, 17(3):259–273, Mar. 1991.
6. B. Berthomieu and F. Vernadat. State class constructions for branching analysis of time Petri nets. In *Proceedings of the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2003)*, volume 2619 of *LNCS*, pages 442–457. Springer, 2003.
7. P. Bouyer, C. Dufourd, E. Fleury, and A. Petit. Updatable timed automata. *Theoretical Computer Science*, 321(2-3):291–345, 2004.
8. F. Cassez and O. H. Roux. Structural translation from time Petri nets to timed automata. *Journal of Systems and Software*, 79(10):1456–1468, 2006.
9. G. Gardey, D. Lime, M. Magnin, and O. H. Roux. Romeo: A tool for analyzing time Petri nets. In *Proceedings of the 17th International Conference on Computer Aided Verification (CAV 2005)*, volume 3576 of *LNCS*, pages 418–423. Springer, 2005.
10. G. Gardey, O. H. Roux, and O. F. Roux. State space computation and analysis of time Petri nets. *Theory and Practice of Logic Programming (TPLP). Special Issue on Specification Analysis and Verification of Reactive Systems*, 6(3):301–320, 2006.
11. GREATSPN web site. http://www.di.unito.it/∼greatspn.
12. KRONOS web site. http://www-verimag.imag.fr/TEMPORISE/kronos/.
13. D. Lime and O. H. Roux. Model checking of time Petri nets using the state class timed automaton. *Journal of Discrete Events Dynamic Systems - Theory and Applications (DEDS)*, 16(2):179–205, 2006.
14. M. Menasche and B. Berthomieu. Time Petri nets for analyzing and verifying time dependent protocols. *Protocol Specification, Testing and Verification III*, pages 161–172, 1983.
15. P. M. Merlin and D. J. Farber. Recoverability of communication protocols: Implications of a theoretical study. *IEEE Trans. Comm.*, 24(9):1036–1043, Sept. 1976.
16. ROMEO web site. http://romeo.rts-software.org/.
17. J. Toussaint, F. Simonot-Lion, and J.-P. Thomesse. Time constraints verification method based on time Petri nets. In *Proceedings of the 6th IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems (FTDCS'97)*, pages 262–267. IEEE Computer Society Press, 1997.
18. UPPAAL web site. http://www.uppaal.com.
19. S. Yovine. Kronos: A verification tool for real-time systems. *International Journal of Software Tools for Technology Transfer*, 1(1/2):123–133, 1997.