Structural translation from time petri nets to timed automata

Franck Cassez, Olivier Henri Roux

Structural Translation from Time Petri Nets to Timed Automata

Franck Cassez and Olivier H. Roux

IRCCyN/CNRS UMR 6597
BP 92101
1 rue de la Noë
44321 Nantes Cedex 3
France

Abstract

In this paper, we consider Time Petri Nets (TPN) where time is associated with transitions. We give a formal semantics for TPNs in terms of Timed Transition Systems. Then, we propose a translation from TPNs to Timed Automata (TA) that preserves the behavioral semantics (timed bisimilarity) of the TPNs. For the theory of TPNs this result is two-fold: i) reachability problems and more generally TCTL model-checking are decidable for bounded TPNs; ii) allowing strict time constraints on transitions for TPNs preserves the results described in i). The practical applications of the translation are: i) one can specify a system using both TPNs and Timed Automata and a precise semantics is given to the composition; ii) one can use existing tools for analyzing timed automata (like KRONOS, UPPAAL or CMC) to analyze TPNs. In this paper we describe the new feature of the tool ROMEO that implements our translation of TPNs in the UPPAAL input format. We also report on experiments carried out on various examples and compare the result of our method to state-of-the-art tool for analyzing TPNs.

1 Introduction

Petri Nets with Time. The two main extensions of Petri Nets with time are Time Petri Nets (TPNs) [21] and Timed Petri Nets [25]. For TPNs a transition can fire within a time interval whereas for Timed Petri Nets it fires as soon as possible. Among Timed Petri Nets, time can be considered relative to places or transitions [27,23]. The two corresponding subclasses namely P-Timed Petri Nets and T-Timed Petri Nets are expressively equivalent [27,23]. The same classes are defined for TPNs i.e. T-TPNs and P-TPNs, but both classes of Timed Petri Nets are included in both P-TPNs and T-TPNs [23]. P-TPNs and T-TPNs are proved to be incomparable in [17]. Finally TPNs form

Preprint submitted to Elsevier Science *Received: date / Revised version: date*

a subclass of Time Stream Petri Nets [14] which were introduced to model multimedia applications.

The class T-TPNs is the most commonly-used subclass of TPNs and in this paper we focus on this subclass that will be henceforth referred to as TPN. For classical TPNs, boundedness is undecidable, and works on this model report undecidability results, or decidability under the assumption that the TPN is bounded (e.g. reachability in [24]). Recent work [1,13] consider timed arc Petri nets where each token has a clock representing his "age". The authors prove that coverability and boundedness are decidable for this class of timed arc Petri nets by applying a backward exploration technique. However, they assume a lazy (non-urgent) behavior of the net: the firing of transitions may be delayed, even if that implies that some transitions are disabled because their input tokens become too old.

Verifying Time Petri Nets. The behavior of a TPN can be defined by timed firing sequences which are sequences of pairs (t, d) where t is a transition of the TPN and $d \in \mathbb{R}_{\geq 0}$. A sequence of transitions like $\omega = (t_1, d_1)$ $(t_2, d_2) \dots (t_n, d_n) \dots$ indicates that t_1 is fired after d_1 time units, then t_2 is fired after d_2 time units, and so on, so that transition t_i is fired at absolute time $\sum_{k=1}^{i} d_k$. A *marking* M is *reachable* in a TPN if there is a timed firing sequence ω from the initial marking M_0 to M. Reachability analysis of TPNs relies on the construction of the so-called States Class Graph (SCG) that was introduced in [6] and later refined in [5]. It has been recently improved in [19] by using partial-order reduction methods.

For bounded TPNs, the SCG construction obviously solves the *marking reachability* problem (Given a marking M, "Can we reach M from M_0?"). If one wants to solve the *state reachability* problem (Given M and $v \in \mathbb{R}_{\geq 0}$ and a transition t, "Can we reach a marking M such that transition t has been enabled for v time units?") the SCG is not sufficient and an alternative graph, the *strong state class* graph is introduced for this purpose in [7]. The two previous graphs allow for checking LTL properties. Another graph can be constructed that preserves CTL* properties. Anyway none of the previous graphs is a good [1] abstraction (accurate enough) for checking *quantitative* real-time properties e.g. "it is not possible to stay in marking M more than n time units" or "from marking M, marking M' is always reached within n time units". The two main (efficient) tools used to verify TPNs are Tina [4] and Romeo [15].

Timed Automata. Timed Automata (TA) were introduced by Alur & Dill [2] and have since been extensively studied. This model is an extension of finite automata with (dense time) *clocks* and enables one to specify real-time systems.

[1] The use of *observers* is of little help as it requires to specify a property as a TPN; thus it is hard to specify properties on markings.

It has been shown that model-checking for TCTL properties is decidable [2,16] for TA and some of their extensions [11]. There also exist several efficient tools like UPPAAL [22], KRONOS [30] and CMC [18] for model-checking TA and many real-time industrial applications have been specified and successfully verified with them.

Related Work. The relationship between TPNs and TA has not been much investigated. In [28] J. Sifakis and S. Yovine are mainly concerned with *compositionality* problems. They show that for a subclass of 1-safe Time Stream Petri Nets, the usual notion of composition used for TA is not suitable to describe this type of Petri Nets as the composition of TA. Consequently, they propose Timed Automata with Deadlines and flexible notions of compositions. In [8] the authors consider Petri nets with deadlines (PND) that are 1-safe Petri nets extended with clocks. A PND is a timed automaton with deadlines (TAD) where the discrete transition structure is the corresponding marking graph. The transitions of the marking graph are subject to the same timing constraints as the transitions of the PND. The PND and the TAD have the same number of clocks. They propose a translation of safe TPN into PND with a clock for each input arc of the initial TPN. It defines (by transitivity) a translation of safe TPN into TAD (that can be considered as standard timed automata). In [9] the authors consider an extension of Time Petri Nets (*PRES+*) and propose a translation into hybrid automata. Correctness of the translation is not proved. Moreover the method is defined only for 1-safe nets.

In another line of work, Sava [26] considers bounded TPN where the underlying Petri net is not necessarily safe and proposes an algorithm to translate the TPN into a timed automaton (one clock is needed for each transition of the original TPN). However, the author does not give any proof that this translation is correct (i.e. it preserves some equivalence relation between the semantics of the original TPN and the computed TA) and neither that the algorithm terminates (even if the TPN is bounded).

Lime and Roux proposed an extension in [20] of the state class graph construction that allows to build the state class graph of a bounded TPN as a timed automaton. They prove that this timed automaton and the TPN are timed-bisimilar and they also prove a relative minimality result of the number of clocks needed in the obtained automaton.

The first two approaches are structural but are limited to Petri nets whose underlying net is 1-safe. The last two approaches rely on the computation of the entire (symbolic) state space of the TPN and are limited to bounded TPN. For example in [20], a timed automaton is indeed computed from a TPN but this requires timing information that are collected by computing a State Class Graph. Moreover, if one uses tools for analyzing the timed automaton, the timing correspondence with the TPN we started with is not easy to infer:

the clocks of the timed automaton do not have a "uniform" meaning [2] in the TPN.

In this article, we consider a structural translation from TPN (not necessary bounded) to TA. The drawbacks of the previous translation are thus avoided because we do not need to compute a State Class Graph and have an easy correspondence of the clocks of our TA with the timing constraints on the transitions of the TPN we started with. It also extends previous results in the following directions: first we can easily prove that our translation is correct and terminates as it is a syntactic translation and it produces a timed automaton that is timed bisimilar to the TPN we started with. Notice that the timed automaton contains integer variables that correspond to the marking of the Petri net and that it may have an unbounded number of locations. However timed bisimilarity holds even in the unbounded case. In case the Petri net is bounded we obtain a timed automaton with a finite number of locations and we can check for TCTL properties of the original TPN. Second as it is a structural translation it does not need expensive computation (like the State Class Graph) to obtain a timed automaton. This has a practical application as it enables one to use efficient existing tools for TA to analyze the TPN.

Our Contribution. We first give a formal semantics for Time Petri Nets [21] in terms of Timed Transition Systems. Then we present a structural translation of a TPN into a synchronized product of timed automata that preserves the semantics (in the sense of *timed bisimilarity*) of the TPN. This yields theoretical and practical applications of this translation : i) TCTL [2,16] model-checking is decidable for bounded TPNs and TCTL properties can now be checked (efficiently) for TPNs with existing tools for analyzing timed automata (like KRONOS, UPPAAL or CMC); ii) allowing strict time constraints on transitions for TPNs preserves the previous result : this leads to an extension of the original TPN model for which TCTL properties can be decided; iii) one can specify a system using both TPNs and Timed Automata and a precise semantics is given to the composition; iv) as the translation is structural, one can use unboundedness testing methods to detect behavior leading to the unboundedness of a TPN.

Some of the above mentioned results appeared in the preliminary version of this paper [12]. In this extended version we have added the proofs of some theorems and more importantly a section that deals with the implementation of our approach: i) we have implemented the structural translation of TPNs into UPPAAL TAs in the tool ROMEO; ii) we have compared our approach to

[2] Assume you manage to prove that a property P is false on the timed automaton associated with a TPN and this happens because clock x (in the TA) has a particular value. There is no easy way of feeding this information back in terms of timing constraints on the transitions of the TPN.

existing tools for analyzing TPNs.

Outline of the paper. Section 2 introduces the semantics of TPNs in terms of timed transition systems and the basics of TA. In Section 3 we show how to build a synchronized product of TA that is timed bisimilar to a TPN. We show how it enables us to check for real-time properties expressed in TCTL in Section 4. In section 5 we describe the implementation of our translation in the tool ROMEO, and report on experiments we have carried out with UPPAAL to check properties of TPNs. Finally we conclude with our ongoing work and perspectives in Section 6.

2 Time Petri Nets and Timed Automata

Notations. We denote by B^A the set of mappings from A to B. If A is finite and $|A| = n$, an element of B^A is also a vector in B^n. The usual operators $+, -, <$ and $=$ are used on vectors of A^n with $A = \mathbb{N}, \mathbb{Q}, \mathbb{R}$ and are the pointwise extensions of their counterparts in A. For a *valuation* $\nu \in A^n, d \in A$, $\nu + d$ denotes the vector $(\nu + d)_i = \nu_i + d$, and for $A' \subseteq A$, $\nu[A' \mapsto 0]$ denotes the valuation ν' with $\nu'(x) = 0$ for $x \in A'$ and $\nu'(x) = \nu(x)$ otherwise. We denote $\mathcal{C}(X)$ for the *simple constraints* over a set of variables X. $\mathcal{C}(X)$ is defined to be the set of boolean combinations (with the connectives $\{\wedge, \vee, \neg\}$) of terms of the form $x - x' \bowtie c$ or $x \bowtie c$ for $x, x' \in X$ and $c \in \mathbb{N}$ and $\bowtie \in \{<, \leq, =, \geq, >\}$. Given a formula $\varphi \in \mathcal{C}(X)$ and a valuation $\nu \in A^n$, we denote by $\varphi(\nu)$ the truth value obtained by substituting each occurrence of x in φ by $\nu(x)$. For a transition system we write transitions as $s \xrightarrow{a} s'$ and a sequence of transitions of the form $s_0 \xrightarrow{a_1} s_1 \rightarrow \cdots \xrightarrow{a_n} s_n$ as $s_0 \stackrel{w}{\Longrightarrow} s_n$ with $w = a_1 a_2 \cdots a_n$.

2.1 Time Petri Nets

The model. Time Petri Nets were introduced in [21] and extend Petri Nets with timing constraints on the firings of transitions.

Definition 1 (Time Petri Net) *A* Time Petri Net $\mathcal{T}$ *is a tuple* $(P, T, {}^\bullet(.), (.)^\bullet, M_0, (\alpha, \beta))$ *where:* $P = \{p_1, p_2, \cdots, p_m\}$ *is a finite set of* places *and* $T = \{t_1, t_2, \cdots, t_n\}$ *is a finite set of* transitions*;* ${}^\bullet(.) \in (\mathbb{N}^P)^T$ *is the* backward *incidence mapping;* $(.)^\bullet \in (\mathbb{N}^P)^T$ *is the* forward *incidence mapping;* $M_0 \in \mathbb{N}^P$ *is the* initial *marking;* $\alpha \in (\mathbb{Q}_{\geq 0})^T$ *and* $\beta \in (\mathbb{Q}_{\geq 0} \cup \{\infty\})^T$ *are respectively the* earliest *and* latest *firing time mappings.*

Semantics of Time Petri Nets. The semantics of TPNs can be given in term of Timed Transition Systems (TTS) which are usual transition systems with two types of labels: discrete labels for events and positive reals labels for time elapsing.

$\nu \in (\mathbb{R}_{\geq 0})^n$ is a *valuation* such that each value ν_i is the elapsed time since the last time transition t_i was enabled. $\mathbf{0}$ is the initial valuation with $\forall i \in [1..n], \mathbf{0}_i = 0$. A *marking* M of a TPN is a mapping in $\mathbb{N}^P$ and if $M \in \mathbb{N}^P$, $M(p_i)$ is the number of tokens in place p_i. A transition t is *enabled* in a marking M iff $M \geq {}^\bullet t$. The predicate $\uparrow enabled(t_k, M, t_i) \in \mathbb{B}$ is true if t_k is enabled by the firing of transition t_i from marking M, and false otherwise. This definition of enabledness is based on [5,3] which is the most common one. In this framework, a transition t_k is *newly enabled* after firing t_i from marking M if "it is not enabled by $M - {}^\bullet t_i$ and is enabled by $M' = M - {}^\bullet t_i + t_i{}^\bullet$" [5].

Formally this gives:

$$\uparrow enabled(t_k, M, t_i) = \Big(M - {}^\bullet t_i + t_i{}^\bullet \geq {}^\bullet t_k\Big) \wedge \Big((M - {}^\bullet t_i < {}^\bullet t_k) \vee (t_k = t_i)\Big) \quad (1)$$

Definition 2 (Semantics of TPN) *The semantics of a TPN $\mathcal{T}$ is a timed transition system $S_\mathcal{T} = (Q, q_0, \rightarrow)$ where: $Q = \mathbb{N}^P \times (\mathbb{R}_{\geq 0})^n$, $q_0 = (M_0, \mathbf{0})$, $\longrightarrow \in Q \times (T \cup \mathbb{R}_{\geq 0}) \times Q$ consists of the discrete and continuous transition relations:*

- *the discrete transition relation is defined for all $t_i \in T$ by $(M, \nu) \xrightarrow{t_i} (M', \nu')$ iff:*
$$\begin{cases} M \geq {}^\bullet t_i \wedge M' = M - {}^\bullet t_i + t_i{}^\bullet \\ \alpha(t_i) \leq \nu_i \leq \beta(t_i) \\ \nu'_k = \begin{cases} 0 & \text{if } \uparrow enabled(t_k, M, t_i), \\ \nu_k & \text{otherwise.} \end{cases} \end{cases}$$
- *the continuous transition relation is defined for all $d \in \mathbb{R}_{\geq 0}$ by $(M, \nu) \xrightarrow{\epsilon(d)} (M, \nu')$ iff:*
$$\begin{cases} \nu' = \nu + d \\ \forall k \in [1..n], \Big(M \geq {}^\bullet t_k \Longrightarrow \nu'_k \leq \beta(t_k)\Big) \end{cases}$$

A run *of a time Petri net $\mathcal{T}$ is a (finite or infinite) path in $S_\mathcal{T}$ starting in q_0. The set of runs of $\mathcal{T}$ is denoted by $[\![\mathcal{T}]\!]$. The set of* reachable markings *of $\mathcal{T}$ is denoted $Reach(\mathcal{T})$. If the set $Reach(\mathcal{T})$ is finite we say that $\mathcal{T}$ is* bounded*. As a shorthand we write $(M, \nu) \longrightarrow_e^d (M', \nu')$ for a sequence of time elapsing and discrete steps like $(M, \nu) \xrightarrow{\epsilon(d)} (M'', \nu'') \xrightarrow{e} (M', \nu')$.*

This definition may need some comments. Our semantics is based on the common definition of [5,3] for safe TPNs.

First, previous formal semantics [5,19,23,3] for TPNs usually require the TPNs to be *safe*. Our semantics encompasses the whole class of TPNs and is fully consistent with the previous semantics when restricted to safe TPNs [3]. Thus, we have given a semantics to multiple enabledness of transitions which seems the most simple and adequate. Indeed, several interpretations can be given to multiple enabledness [5].

Second, some variations can be found in the literature about TPNs concerning the firing of transitions. The paper [23] considers two distinct semantics: Weak Time Semantics (WTS) and Strong Time Semantics (STS). According to WTS, a transition *can* be fired only in its time interval whereas in STS, a transition *must* fire within its firing interval unless disabled by the firing of others. The most commonly used semantics is STS as in [21,5,23,3].

Third, it is possible for the TPN to be zeno or unbounded. In the case it is unbounded, the discrete component of the state space of the timed transition system is infinite. If $\forall i, \alpha(t_i) > 0$ then the TPN is non-zeno and the requirement that time diverges on each run is fulfilled. Otherwise, if the TPN is bounded and at least one lower bound is 0, the zeno or non-zeno property can be decided [16] for the TPN using the equivalent timed automaton we build in section 3.

2.2 Timed Automata and Products of Timed Automata

Timed automata [2] are used to model systems which combine *discrete* and *continuous* evolutions.

Definition 3 (Timed Automaton) *A* Timed Automaton *H is a tuple (N, l_0, X, A, E, Inv) where: N is a finite set of* locations*; $l_0 \in N$ is the* initial location*; X is a finite set of positive real-valued* clocks*; A is a finite set of* actions*; $E \subseteq N \times \mathcal{C}(X) \times A \times 2^X \times N$ is a finite set of* edges*, $e = \langle l, \gamma, a, R, l' \rangle \in E$ represents an edge from the location l to the location l' with the guard γ, the label a and the reset set $R \subseteq X$; and $Inv \in \mathcal{C}(X)^N$ assigns an* invariant *to any location. We restrict the invariants to conjuncts of terms of the form $c \leq r$ for $c \in X$ and $r \in \mathbb{N}$.*

The semantics of a timed automaton is also a timed transition system.

Definition 4 (Semantics of a TA) *The semantics of a timed automaton $H = (N, l_0, X, A, E, Inv)$ is a timed transition system $S_H = (Q, q_0, \rightarrow)$ with*

[3] If we accept the difference with [19] in the definition of the reset instants for newly enabled transitions.

$Q = N \times (\mathbb{R}_{\leq 0})^X$, $q_0 = (l_0, \mathbf{0})$ *is the initial state and* $\rightarrow$ *consists of the discrete and continuous transition relations:*

- *the discrete transition relation if defined for all* $a \in A$ *by* $(l, v) \xrightarrow{a} (l', v')$ *if:*

$$\exists\, (l, \gamma, a, R, l') \in E \text{ s.t. } \begin{cases} \gamma(v) = \mathit{tt}, \\ v' = v[R \mapsto 0] \\ Inv(l')(v') = \mathit{tt} \end{cases}$$

- *the continuous transitions is defined for all* $t \in \mathbb{R}_{\geq 0}$ *by* $(l, v) \xrightarrow{\epsilon(t)} (l', v')$ *if:*

$$\begin{cases} l = l' \qquad v' = v + t \quad \textit{and} \\ \forall\, 0 \leq t' \leq t,\ Inv(l)(v + t') = \mathit{tt} \end{cases}$$

A run *of a timed automaton* H *is a path in* S_H *starting in* q_0*. The set of runs of* H *is denoted by* $[\![H]\!]$*.*

Product of Timed Automata. It is convenient to describe a system as a parallel composition of timed automata. To this end, we use the classical composition notion based on a *synchronization function* à la Arnold-Nivat. Let $X = \{x_1, \cdots, x_n\}$ be a set of clocks, H_1, ..., H_n be n timed automata with $H_i = (N_i, l_{i,0}, X, A, E_i, Inv_i)$. A *synchronization function* f is a partial function from $(A \cup \{\bullet\})^n \hookrightarrow A$ where $\bullet$ is a special symbol used when an automaton is not involved in a step of the global system. Note that f is a synchronization function with renaming. We denote by $(H_1 | \ldots | H_n)_f$ the parallel composition of the H_i's w.r.t. f. The configurations of $(H_1 | \ldots | H_n)_f$ are pairs $(\mathbf{l}, \mathbf{v})$ with $\mathbf{l} = (l_1, \ldots, l_n) \in N_1 \times \ldots \times N_n$ and $\mathbf{v} = (v_1, \cdots, v_n)$ where each v_i is the value of the clock $x_i \in X$. Then the semantics of a synchronized product of timed automata is also a timed transition system: the synchronized product can do a discrete transition if all the components agree to and time can progress in the synchronized product also if all the components agree to. This is formalized by the following definition:

Definition 5 (Semantics of a Product of TA) *Let* $H_1, \ldots, H_n$ *be* n *timed automata with* $H_i = (N_i, l_{i,0}, X, A, E_i, Inv_i)$*, and* f *a (partial) synchronization function* $(A \cup \{\bullet\})^n \hookrightarrow A$*. The semantics of* $(H_1 | \ldots | H_n)_f$ *is a timed transition system* $S = (Q, q_0, \rightarrow)$ *with* $Q = N_1 \times \ldots \times N_n \times (\mathbb{R}_{\geq 0})^X$*,* q_0 *is the initial state* $((l_{1,0}, \ldots, l_{n,0}), \mathbf{0})$ *and* $\rightarrow$ *is defined by:*

- $(\mathbf{l}, \mathbf{v}) \xrightarrow{b} (\mathbf{l}', \mathbf{v}')$ *if there exists* $(a_1, \ldots, a_n) \in (A \cup \{\bullet\})^n$ *s.t.* $f(a_1, \ldots, a_n) = b$ *and for any* i *we have:*
  - *If* $a_i = \bullet$*, then* $\mathbf{l}'[i] = \mathbf{l}[i]$ *and* $\mathbf{v}'[i] = \mathbf{v}[i]$*,*
  - *If* $a_i \in A$*, then* $(\mathbf{l}[i], \mathbf{v}[i]) \xrightarrow{a_i} (\mathbf{l}'[i], \mathbf{v}'[i])$*.*

- $(\mathbf{l}, \mathbf{v}) \xrightarrow{\epsilon(t)} (\mathbf{l}, \mathbf{v}')$ *if for all* $i \in [1..n]$*, every* H_i *agrees on time elapsing* i.e. $(\mathbf{l}[i], \mathbf{v}[i]) \xrightarrow{\epsilon(t)} (\mathbf{l}[i], \mathbf{v}'[i])$.

We could equivalently define the product of n timed automata syntactically, building a new timed automaton from the n initial ones. In the sequel we consider a product $(H_1|\ldots|H_n)_f$ to be a timed automaton the semantics of which is timed bisimilar to the semantics of the product we have given in Definition 5.

3 From Time Petri Nets to Timed Automata

In this section, we build a synchronized product of timed automata from a TPN so that the behaviors of the two are in a one-to-one correspondence.

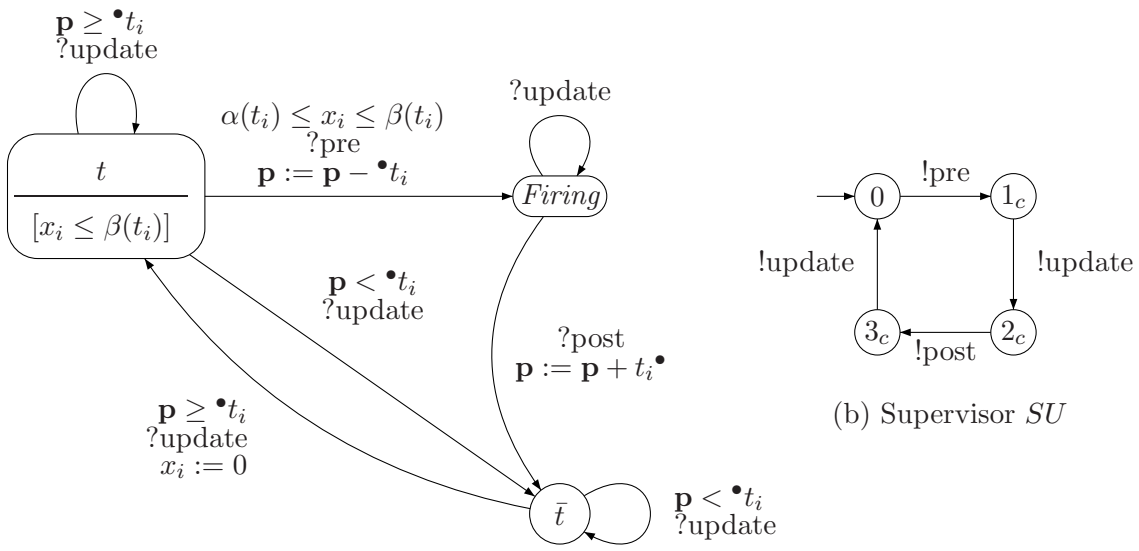
3.1 Translating Time Petri Nets into Timed Automata

We start with a TPN $\mathcal{T} = (P, T, {}^\bullet(.), (.)^\bullet, M_0, (\alpha, \beta))$ with $P = \{p_1, \cdots, p_m\}$ and $T = \{t_1, \cdots, t_n\}$.

Timed Automaton for one Transition. We define one timed automaton $\mathcal{A}_i$ for each transition t_i of T (see Fig. 1.a). This timed automaton has one clock x_i. Also the states of the automaton $\mathcal{A}_i$ give the state of the transition t_i: in state t the transition is enabled; in state $\bar{t}$ it is disabled and in *Firing* it is being fired. The initial state of each $\mathcal{A}_i$ depends on the initial marking M_0 of the Petri net we want to translate. If $M_0 \geq {}^\bullet t_i$, then the initial state is t otherwise it is $\bar{t}$. This automaton updates an array of integers $\mathbf{p}$ (s.t. $\mathbf{p}[i]$ is the number of tokens in place p_i) which is shared by all the $\mathcal{A}_i$'s. This is not covered by Definition 5, which is very often extended ([22]) with integer variables (this does not affect the expressiveness of the model when the variables are bounded).

The Supervisor. The automaton for the supervisor SU is depicted on Fig. 1.b. The locations 1 to 3 subscripted with a "c" are assumed to be urgent or committed[4] which means that no time can elapse while visiting them. We denote by $\Delta(\mathcal{T}) = (SU \mid \mathcal{A}_1 \mid \cdots \mid \mathcal{A}_n)_f$ the timed automaton associated to the TPN $\mathcal{T}$. The supervisor's initial state is 0. Let us define the synchronization

[4] In SU, *committed* locations can be simulated by adding an extra variable: see [29] Appendix A for details.



(a) The automaton $\mathcal{A}_i$ for transition t_i

(b) Supervisor SU

Fig. 1. Automata for the Transitions and the Supervisor

function[5] f with $n+1$ parameters defined by:

- $f(!pre, \bullet, \cdots, ?pre, \bullet, \cdots) = pre_i$ if $?pre$ is the $(i+1)$th argument and all the other arguments are $\bullet$,
- $f(!post, \bullet, \cdots, ?post, \bullet, \cdots) = post_i$ if $?post$ is the $(i+1)$th argument and all the other arguments are $\bullet$,
- $f(!update, ?update, \cdots, ?update) = update$.

We will prove in the next subsection that the semantics of $\Delta(\mathcal{T})$ is closely related to the semantics of $\mathcal{T}$. For this we have to relate the states of $\mathcal{T}$ to the states of $\Delta(\mathcal{T})$ and we define the following equivalence:

Definition 6 (State Equivalence) *Let (M, ν) and $((s, \mathbf{p}), \mathbf{q}, \mathbf{v})$ be, respectively, a state of $S_\mathcal{T}$ and a configuration[6]. Then $(M, \nu) \approx ((s, \mathbf{p}), \mathbf{q}, \mathbf{v})$ if:*

$$\begin{cases} s = 0, \\ \forall i \in [1..m], \quad \mathbf{p}[i] = M(p_i), \\ \forall k \in [1..n], \quad \mathbf{q}[k] = \begin{cases} t \text{ if } M \geq {}^\bullet t_k, \\ \bar{t} \text{ otherwise} \end{cases} \\ \forall k \in [1..n], \quad \mathbf{v}[k] = \nu_k \end{cases}$$

[5] The first element of the vector refers to the supervisor move.

[6] $(s, \mathbf{p}) \in \{0, 1_c, 2_c, 3_c\} \times \mathbb{N}^m$ is the state of SU, $\mathbf{q}$ gives the product location of $\mathcal{A}_1 \mid \cdots \mid \mathcal{A}_n$, and $\mathbf{v}[i], i \in [1..n]$ gives the value of the clock x_i.

We now prove that our translation preserves the behaviors of the initial TPN in the sense that the semantics of the TPN and its translation are timed bisimilar. We assume a TPN $\mathcal{T}$ and $S_{\mathcal{T}} = (Q, q_0, \rightarrow)$ its semantics. Let $\mathcal{A}_i$ be the automaton associated with transition t_i of $\mathcal{T}$ as described by Fig. 1.a, SU the supervisor automaton of Fig. 1.b and f the synchronization function defined previously. The semantics of $\Delta(\mathcal{T}) = (SU \mid \mathcal{A}_1 \mid \cdots \mid \mathcal{A}_n)_f$ is the TTS $S_{\Delta(\mathcal{T})} = (Q_{\Delta(\mathcal{T})}, q_0^{\Delta(\mathcal{T})}, \rightarrow)$.

Theorem 1 (Timed Bisimilarity) *For $(M,\nu) \in S_{\mathcal{T}}$ and $((0,\mathbf{p}),\mathbf{q},\mathbf{v}) \in S_{\Delta(\mathcal{T})}$ such that $(M,\nu) \approx ((0,\mathbf{p}),\mathbf{q},\mathbf{v})$ the following holds:*

$$(M,\nu) \xrightarrow{t_i} (M',\nu') \text{ iff } \begin{cases} ((0,\mathbf{p}),\mathbf{q},\mathbf{v}) \stackrel{w_i}{\Longrightarrow} ((0,\mathbf{p}'),\mathbf{q}',\mathbf{v}') \text{ with} \\ w_i = pre_i.update.post_i.update \text{ and} \\ (M',\nu') \approx ((0,\mathbf{p}'),\mathbf{q}',\mathbf{v}') \end{cases} \quad (2)$$

$$(M,\nu) \xrightarrow{\epsilon(d)} (M',\nu') \text{ iff } \begin{cases} ((0,\mathbf{p}),\mathbf{q},\mathbf{v}) \xrightarrow{\epsilon(d)} ((0,\mathbf{p}'),\mathbf{q}',\mathbf{v}') \text{ and} \\ (M',\nu') \approx ((0,\mathbf{p}'),\mathbf{q}',\mathbf{v}') \end{cases} \quad (3)$$

Proof 1 *We first prove statement (2). Assume $(M,\nu) \approx ((0,\mathbf{p}),\mathbf{q},\mathbf{v})$. Then as t_i can be fired from (M,ν) we have: (i) $M \geq {}^\bullet t_i$, (ii) $\alpha(t_i) \leq \nu_i \leq \beta(t_i)$, (iii) $M' = M - {}^\bullet t_i + t_i^\bullet$, and (iv) $\nu'_k = 0$ if $\uparrow enabled(t_k, M, t_i)$ and $\nu'_k = \nu_k$ otherwise. From (i) and (ii) and the state equivalence we deduce that $\mathbf{q}[i] = t$ and $\alpha(t_i) \leq \mathbf{v}[i] \leq \beta(t_i)$. Hence ?pre is enabled in $\mathcal{A}_i$. In state 0 for the supervisor, !pre is the only possible transition. As the synchronization function f allows $(!pre, \bullet, \cdots, ?pre, \cdots, \bullet)$ the global action pre_i is possible. After this move $\Delta(\mathcal{T})$ reaches state $((1,\mathbf{p_1}),\mathbf{q_1},\mathbf{v_1})$ such that for all $k \in [1..n]$, $\mathbf{q_1}[k] = \mathbf{q}[k], \forall k \neq i$ and $\mathbf{q_1}[i] = Firing$. Also $\mathbf{p_1} = \mathbf{p} - {}^\bullet t_i$ and $\mathbf{v_1} = \mathbf{v}$.*

Now the only possible transition when the supervisor is in state 1 is an update transition where all the $\mathcal{A}_i$'s synchronize according to f. From $((1,\mathbf{p_1}),\mathbf{q_1},\mathbf{v_1})$ we reach $((2,\mathbf{p_2}),\mathbf{q_2},\mathbf{v_2})$ with $\mathbf{p_2} = \mathbf{p_1}$, $\mathbf{v_2} = \mathbf{v_1}$. For all $k \in [1..n]$, $k \neq i$, $\mathbf{q_2}[k] = t$ if $\mathbf{p_1} \geq {}^\bullet t_k$ and $\mathbf{q_2}[k] = \bar{t}$ otherwise. Also $\mathbf{q_2}[i] = Firing$. The next global transition must be a $post_i$ transition leading to $((3,\mathbf{p_3}),\mathbf{q_3},\mathbf{v_3})$ with $\mathbf{p_3} = \mathbf{p_2} + t_i^\bullet$, $\mathbf{v_3} = \mathbf{v_2}$ and for all $k \in [1..n]$, $\mathbf{q_3}[k] = \mathbf{q_2}[k], \forall k \neq i$ and $\mathbf{q_3}[i] = \bar{t}$.

From this last state only an update transition leading to $((0,\mathbf{p_4}),\mathbf{q_4},\mathbf{v_4})$ is allowed, with $\mathbf{p_4} = \mathbf{p_3}$, $\mathbf{v_4}$ and $\mathbf{q_4}$ given by: for all $k \in [1..n]$, $\mathbf{q_4}[k] = t$ if $\mathbf{p_3} \geq {}^\bullet t_k$ and $\bar{t}$ otherwise. $\mathbf{v_4}[k] = 0$ if $\mathbf{q_3}[k] = \bar{t}$ and $\mathbf{q_4}[k] = t$ and $\mathbf{v_4}[k] = \mathbf{v_1}[k]$ otherwise. We then just notice that $\mathbf{q_3}[k] = \bar{t}$ iff $\mathbf{p} - {}^\bullet t_i < {}^\bullet t_k$ and $\mathbf{q_4}[k] = t$ iff $\mathbf{p} - {}^\bullet t_i + t_i^\bullet \geq {}^\bullet t_k$. This entails that $\mathbf{v_4}[k] = 0$ iff $\uparrow enabled(t_k, \mathbf{p}, t_i)$ and with (iv)

gives $\nu'_k = \mathbf{v_4}[k]$. *As* $\mathbf{p_4} = \mathbf{p_3} = \mathbf{p_2} + t_i^\bullet = \mathbf{p_1} - {}^\bullet t_i + t_i^\bullet = \mathbf{p} - {}^\bullet t_i + t_i^\bullet$ *using (iii) we have* $\forall i \in [1..m], M'(p_i) = \mathbf{p_4}[i]$. *Hence we conclude that* $((0, \mathbf{p_4}), \mathbf{q_4}, \mathbf{v_4}) \approx (M', \nu')$.

The converse of statement (2) is straightforward following the same steps as the previous ones.

We now focus on statement (3). According to the semantics of TPNs, a continuous transition $(M, \nu) \xrightarrow{\epsilon(d)} (M', \nu')$ *is allowed iff* $\nu = \nu' + d$ *and* $\forall k \in [1..n], (M \geq {}^\bullet t_k \Longrightarrow \nu'_k \leq \beta(t_k))$. *As* $(M, \nu) \approx ((0, \mathbf{p}), \mathbf{q}, \mathbf{v})$, *if* $M \geq {}^\bullet t_k$ *then* $\mathbf{q}[k] = t$ *and the continuous evolution for* $\mathcal{A}_k$ *is constrained by the invariant* $x_k \leq \beta(t_k)$. *Otherwise* $\mathbf{q}[k] = \bar{t}$ *and the continuous evolution is unconstrained for* $\mathcal{A}_k$. *No constraints apply for the supervisor in state* 0. *Hence the result.* □

We can now state a useful corollary which enables us to do TCTL model-checking for TPNs in the next section. We write $\Delta((M, \nu)) = ((0, \mathbf{p}), \mathbf{q}, \mathbf{v})$ if $(M, \nu) \approx ((0, \mathbf{p}), \mathbf{q}, \mathbf{v})$, $\Delta(t_i) = pre_i.update.post_i.update$ and also $\Delta(\epsilon(d)) = \epsilon(d)$. Just notice that Δ is one-to-one and we can use Δ^{-1} as well. Then we extend Δ to transitions as: $\Delta((M, \nu) \xrightarrow{e} (M', \nu')) = \Delta((M, \nu)) \xrightarrow{\Delta(e)} \Delta((M', \nu'))$ with $e \in T \cup \mathbb{R}_{\geq 0}$ (as $\Delta(t_i)$ is a word, this transition is a four step transition in $\Delta(\mathcal{T})$). Again we can extend Δ to runs: if $\rho \in [\![\mathcal{T}]\!]$ we denote $\Delta(\rho)$ the associated run in $[\![\Delta(\mathcal{T})]\!]$. Notice that Δ^{-1} is only defined for runs σ of $[\![\Delta(\mathcal{T})]\!]$, the last state of which is of the form $((0, \mathbf{p}), \mathbf{q}, \mathbf{v})$ where the supervisor is in state 0. We denote this property $last(\sigma) \models SU.0$.

Corollary 1 $\Big(\rho \in [\![\mathcal{T}]\!] \wedge \sigma = \Delta(\rho)\Big)$ *iff* $\Big(\sigma \in [\![\Delta(\mathcal{T})]\!] \wedge last(\sigma) \models SU.0\Big)$.

Proof 2 *The proof is a direct consequence of Theorem 1. It suffices to notice that all the finite runs of* $\Delta(\mathcal{T})$ *are of the form*

$$\sigma = (s_0, v_0) \xrightarrow{\delta_1} (s'_0, v'_0) \xrightarrow{w_1} (s_1, v_1) \cdots \quad \xrightarrow{\delta_n} (s'_{n-1}, v'_{n-1}) \xrightarrow{w_n} (s_n, v_n) \qquad (4)$$

with $w_i = pre_i.update.post_i.update$, $\delta_i \in \mathbb{R}_{\geq 0}$, *and using Theorem 1, if* $last(\sigma) \models SU.0$, *there exists a corresponding run* ρ *in* $\mathcal{T}$ *s.t.* $\sigma = \Delta(\rho)$. □

This property will be used in Section 4 when we address the problem of model-checking TCTL for TPNs.

4 TCTL Model-Checking for Time Petri Nets

We can now define TCTL [16] for TPNs. The only difference with the versions of [16] is that the atomic propositions usually associated to states are prop-

erties of markings. For practical applications with model-checkers we assume that the TPNs we check are bounded.

TCTL for TPNs.

Definition 7 (TCTL for TPN) *Assume a TPN with n places, and m transitions $T = \{t_1, t_2, \cdots, t_m\}$. The temporal logic TPN-TCTL is inductively defined by:*

$$\begin{aligned} \textit{TPN-TCTL} ::= & \mathbf{M} \bowtie \bar{V} \mid \mathbf{false} \mid t_k + c \leq t_j + d \mid \neg\varphi \\ & \mid \varphi \rightarrow \psi \mid \varphi \, \exists\mathcal{U}_{\bowtie c} \, \psi \mid \varphi \, \forall\mathcal{U}_{\bowtie c} \, \psi \end{aligned} \tag{5}$$

where $\mathbf{M}$ *and* **false** *are keywords,* $\varphi, \psi \in$ *TPN-TCTL,* $t_k, t_j \in T$*,* $c, d \in \mathbb{Z}$*,* $\bar{V} \in (\mathbb{N} \cup \{\infty\})^n$ *and*[7] $\bowtie \in \{<, \leq, =, >, \geq\}$*.*

Intuitively the meaning of $\mathbf{M} \bowtie \bar{V}$ is that the current marking vector is in relation $\bowtie$ with $\bar{V}$. The meaning of the other operators is the usual one. We use the familiar shorthands $\mathbf{true} = \neg\mathbf{false}$, $\exists\Diamond_{\bowtie c}\phi = \mathbf{true}\,\exists\mathcal{U}_{\bowtie c}\,\phi$ and $\forall\Box_{\bowtie c} = \neg\exists\Diamond_{\bowtie c}\neg\phi$.

The semantics of TPN-TCTL is defined on timed transition systems. Let $\mathcal{T} = (P, T, {}^\bullet(.), (.)^\bullet, M_0, (\alpha, \beta))$ be a TPN and $S_\mathcal{T} = (Q, q_0, \rightarrow)$ the semantics of $\mathcal{T}$. Let $\sigma = (s_0, \nu_0) \longrightarrow^{d_1}_{a_1} \cdots \longrightarrow^{d_n}_{a_n} (s_n, \nu_n) \in [\![\mathcal{T}]\!]$. The truth value of a formula φ of TPN-TCTL for a state (M, ν) is given in Fig. 2.

The TPN $\mathcal{T}$ satisfies the formula φ of TPN-TCTL, which is denoted by $\mathcal{T} \models \varphi$, iff the first state of $S_\mathcal{T}$ satisfies φ, i.e. $(M_0, \mathbf{0}) \models \varphi$.

We will see that thanks to Corollary 1, model-checking TPNs amounts to model-checking timed automata.

Model-Checking for TPN-TCTL. Let us assume we have to model-check formula φ on a TPN $\mathcal{T}$. Our method consists in using the equivalent timed automaton $\Delta(\mathcal{T})$ defined in Section 3. For instance, suppose we want to check $\mathcal{T} \models \forall\Box_{\leq 3}(\mathbf{M} \geq (1, 2))$. The check means that all the states reached within the next 3 time units will have a marking such that p_1 has more than one token and p_2 more than 2. Actually, this is equivalent to checking $\forall\Box_{\leq 3}(SU.0 \rightarrow (\mathbf{p}[1] \geq 1 \wedge \mathbf{p}[2] \geq 2))$ on the equivalent timed automaton. Notice that $\exists\Diamond_{\leq 3}(\mathbf{M} \geq (1, 2))$ reduces to $\exists\Diamond_{\leq 3}(SU.0 \wedge (\mathbf{p}[1] \geq 1 \wedge \mathbf{p}[2] \geq 2))$. We can then define the translation of a formula in TPN-TCTL to standard TCTL for timed automata: we denote TA-TCTL the logic TCTL for timed automata.

[7] The use of ∞ in $\bar{V}$ allows us to handle comparisons like $M(p_1) \leq 2 \wedge M(p_2) \geq 3$ by writing $\mathbf{M} \leq (2, \infty) \wedge \mathbf{M} \geq (0, 3)$.

$$
\begin{array}{lll}
(M,\nu) \models \mathbf{M} \bowtie \bar{V} & \text{iff} & M \bowtie \bar{V} \\
(M,\nu) \not\models \mathbf{false} & & \\
(M,\nu) \models t_k + c \leq t_j + d & \text{iff} & \nu_k + c \leq \nu_j + d \\
(M,\nu) \models \neg\varphi & \text{iff} & (M,\nu) \not\models \varphi \\
(M,\nu) \models \varphi \rightarrow \psi & \text{iff} & (M,\nu) \models \varphi \text{ implies } (M,\nu) \models \psi \\
(M,\nu) \models \varphi\, \exists\mathcal{U}_{\bowtie c}\, \psi & \text{iff} & \exists \sigma \in [\![\mathcal{T}]\!] \ \textit{such that} \\
 & & \begin{cases} (s_0,\nu_0) = (M,\nu) \\ \forall i \in [1..n], \forall d \in [0,d_i), (s_i, \nu_i + d) \models \varphi \\ \left(\sum_{i=1}^{n} d_i\right) \bowtie c \ \textit{and}\ (s_n, v_n) \models \psi \end{cases} \\
(M,\nu) \models \varphi\, \forall\mathcal{U}_{\bowtie c}\, \psi & \text{iff} & \forall \sigma \in [\![\mathcal{T}]\!] \ \textit{we have} \\
 & & \begin{cases} (s_0,\nu_0) = (M,\nu) \\ \forall i \in [1..n], \forall d \in [0,d_i), (s_i, \nu_i + d) \models \varphi \\ \left(\sum_{i=1}^{n} d_i\right) \bowtie c \ \textit{and}\ (s_n, v_n) \models \psi \end{cases}
\end{array}
$$

Fig. 2. Semantics of TPN-TCTL

Definition 8 (From TPN-TCTL to TA-TCTL) *Let φ be a formula of TPN-TCTL. Then the translation $\Delta(\varphi)$ of φ is inductively defined by:*

$$
\begin{array}{l}
\Delta(\mathbf{M} \bowtie \bar{V}) = \bigwedge_{i=1}^{n} (\mathbf{p}[i] \bowtie \bar{V}_i) \\
\Delta(\mathbf{false}) = \mathbf{false} \\
\Delta(t_k + c \bowtie t_j + d) = x_k + c \bowtie x_j + d \\
\Delta(\neg\varphi) = \neg\Delta(\varphi) \\
\Delta(\varphi \rightarrow \psi) = SU.0 \wedge (\Delta(\varphi) \rightarrow \Delta(\psi)) \\
\Delta(\varphi\, \exists\mathcal{U}_{\bowtie c}\, \psi) = (SU.0 \rightarrow \Delta(\varphi))\, \exists\mathcal{U}_{\bowtie c}\, (SU.0 \wedge \Delta(\psi)) \\
\Delta(\varphi\, \forall\mathcal{U}_{\bowtie c}\, \psi) = (SU.0 \rightarrow \Delta(\varphi))\, \forall\mathcal{U}_{\bowtie c}\, (SU.0 \wedge \Delta(\psi))
\end{array}
$$

$SU.0$ means that the supervisor is in state 0 and the clocks x_k are the ones associated with every transition t_k in the translation scheme.

Theorem 2 *Let $\mathcal{T}$ be a TPN and $\Delta(\mathcal{T})$ the equivalent timed automaton. Let (M,ν) be a state of $S_{\mathcal{T}}$ and $((s,\mathbf{p}),\mathbf{q},\mathbf{v}) = \Delta((M,\nu))$ the equivalent state of $S_{\Delta(\mathcal{T})}$ (i.e. $(M,\nu) \approx ((s,\mathbf{p}),\mathbf{q},\mathbf{v})$). Then $\forall\varphi \in$ TPN-TCTL:*

$$(M,\nu) \models \varphi \textit{ iff } ((s,\mathbf{p}),\mathbf{q},\mathbf{v}) \models \Delta(\varphi).$$

Proof 3 *The proof is done by structural induction on the formula of TPN-TCTL. The cases of $\mathbf{M} \bowtie \bar{V}$, $\mathbf{false}$, $t_k + c \leq t_j + d$, $\neg\varphi$ and $\varphi \rightarrow \psi$ are straightforward. We give the full proof for $\varphi\, \exists\mathcal{U}_{\bowtie c}\, \psi$ (the same proof can be carried out for $\varphi\, \forall\mathcal{U}_{\bowtie c}\, \psi$).*

Only if part. *Assume* $(M,\nu) \models \varphi\, \exists\mathcal{U}_{\bowtie c}\, \psi$. *Then by definition, there is a run* ρ *in* $[\![\mathcal{T}]\!]$ *s.t.:*

$$\rho = (s_0,\nu_0) \longrightarrow_{a_1}^{d_1} (s_1,\nu_1) \cdots \longrightarrow_{a_n}^{d_n} (s_n,\nu_n)$$

and $(s_0,\nu_0) = (M,\nu)$, $\sum_{i=1}^{n} d_i \bowtie c$, $\forall i \in [1..n], \forall d \in [0,d_i), (s_i,\nu_i+d) \models \varphi$ *and* $(s_n,\nu_n) \models \psi$. *With corollary 1, we conclude that there is a run* $\sigma = \Delta(\rho)$ *in* $[\![S_{\Delta(\mathcal{T})}]\!]$ *s.t.*

$$\sigma = ((l_0,p_0),\bar{q}_0,v_0)) \Longrightarrow_{w_1}^{d_1} ((l_1,p_1),\bar{q}_1,v_1)) \cdots \quad \cdots \Longrightarrow_{w_n}^{d_n} ((l_n,p_n),\bar{q}_n,v_n))$$

and $\forall i \in [1..n], ((l_i,p_i),\bar{q}_i,v_i)) \approx (s_i,\nu_i)$ *(this entails that* $l_i = 0$.*)*

Since $(s_n,\nu_n) \approx ((l_n,p_n),\bar{q}_n,v_n))$, *using the induction hypothesis on* ψ, *we can assume* $(s_n,\nu_n) \models \psi$ *iff* $((l_n,p_n),\bar{q}_n,v_n)) \models \Delta(\psi)$ *and thus we can conclude that* $((l_n,p_n),\bar{q}_n,v_n)) \models \Delta(\psi)$. *Moreover as* $l_n = 0$ *we have* $((l_n,p_n),\bar{q}_n,v_n)) \models SU.0 \wedge \Delta(\psi)$. *It remains to prove that all intermediate states satisfy* $SU.0 \rightarrow \Delta(\varphi)$. *Just notice that all the intermediate states in* σ *not satisfying* $SU.0$ *between* $((l_i,p_i),\bar{q}_i,v_i)$ *and* $(((l_{i+1},p_{i+1}),q_{i+1}^{-},v_{i+1}))$ *satisfy* $SU.0 \rightarrow \Delta(\psi)$. *Then we just need to prove that the intermediate states satisfying* $SU.0$, *i.e. the states* $((l_i,p_i),\bar{q}_i,v_i)$ *satisfy* $\Delta(\varphi)$. *As for all* $i \in [1..n]$, *we have* $((l_i,p_i),\bar{q}_i,v_i)) \approx (s_i,\nu_i)$, *with the induction hypothesis on* φ, *we have* $\forall i \in [1..n], ((l_i,p_i),\bar{q}_i,v_i)) \models \Delta(\varphi)$. *Moreover, again applying theorem 1, we obtain for all* $d \in [0,d_i)$*:* $((l_i,p_i),\bar{q}_i,v_i{+}d)) \approx (s_i,\nu_i{+}d)$*; applying the induction hypothesis again we conclude that for all* $d \in [0,d_i)$ $((l_i,p_i),\bar{q}_i,v_i{+}d)) \models \Delta(\varphi)$. *Hence* $((l_0,p_0),\bar{q}_0,v_0)) \models (SU.0 \rightarrow \varphi)\, \exists\mathcal{U}_{\bowtie c}\, (SU.0 \wedge \psi)$.

If part. *Assume* $((l_0,p_0),\bar{q}_0,v_0)) \models (SU.0 \rightarrow \Delta(\varphi))\, \exists\mathcal{U}_{\bowtie c}\ (SU.0 \wedge \Delta(\psi))$. *Then there is a run*

$$\sigma = ((l_0,p_0),\bar{q}_0,v_0)) \Longrightarrow_{w_1}^{d_1} ((l_1,p_1),\bar{q}_1,v_1)) \cdots \quad \cdots \Longrightarrow_{w_n}^{d_n} ((l_n,p_n),\bar{q}_n,v_n))$$

with $((l_n,p_n),\bar{q}_n,v_n)) \models SU.0 \wedge \Delta(\psi)$ *and* $\forall i \in [1..n], \forall d \in [0,d_i), ((l_i,p_i),\bar{q}_i,v_i)) \models (SU.0 \rightarrow \Delta(\varphi))$. *As* $((l_n,p_n),\bar{q}_n,v_n)) \models SU.0$, *we can use corollary 1 and we know there exists a run in* $[\![\mathcal{T}]\!]$

$$\rho = \Delta^{-1}(\sigma) = (s_0,\nu_0) \rightarrow_{a_1}^{d_1} (s_1,\nu_1) \cdots \rightarrow_{a_n}^{d_n} (s_n,\nu_n)$$

with $\forall i \in [1..n], ((l_i,p_i),\bar{q}_i,v_i)) \approx (s_i,\nu_i)$. *The induction hypothesis on* $SU.0 \wedge \Delta(\psi)$ *and* $((l_n,p_n),\bar{q}_n,v_n)) \models \mathcal{S}.0 \wedge \Delta(\psi)$ *implies* $(s_n,\nu_n) \models \psi$. *For all the intermediate states of* ρ *we also apply the induction hypothesis: each* $((l_i,p_i),\bar{q}_i,v_i))$ *is equivalent to* (s_i,ν_i) *and all the states* $(s_i,\nu_i+d), d \in [0,d_i)$ *satisfy* φ. *Hence* $(s_0,\nu_0) \models \varphi\, \exists\mathcal{U}_{\bowtie c}\, \psi$. □

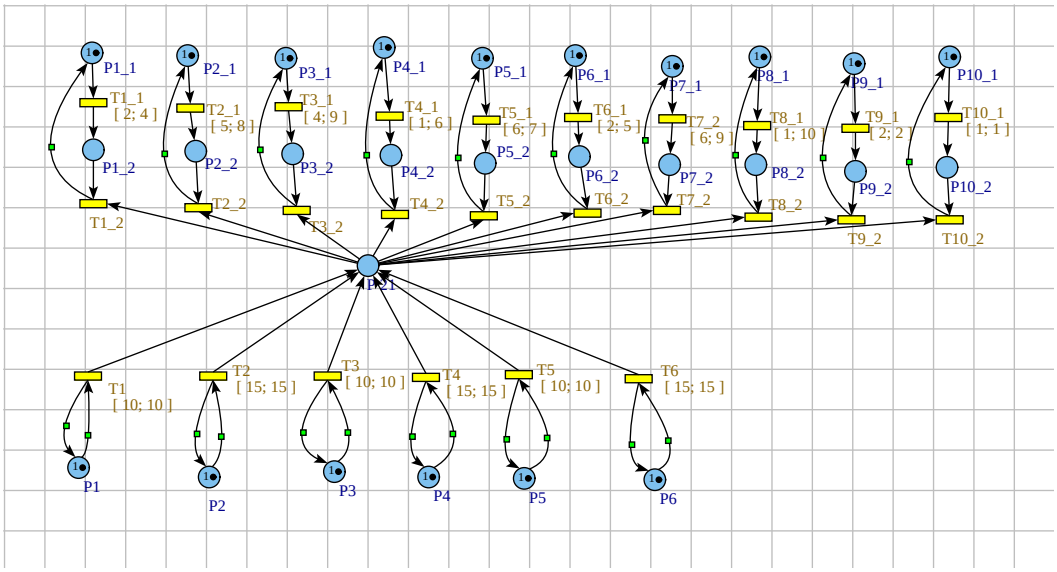


Fig. 3. A TPN for a Producer/Consumer example in Romeo

5 Implementation

In this section we describe some properties of our translation and important implementation details. Then we report on examples we have checked using our approach and Uppaal.

5.1 Translation to Uppaal *input format*

The first step in using our approach is to translate an existing TPN into a product of TA. For this we use the tool Romeo [15] that has been developed for the analysis of TPNs (state space computation and "on the fly" model-checking of reachability properties with a zone-based forward method and with the State Class Graph method). Romeo has a GUI (see Fig. 3) to "draw" Timed Petri Nets and now an *export to* Uppaal feature that implements our translation of a TPN into the equivalent TA in Uppaal input format [8].

The textual input format for TPN in Romeo is XML and the timed automaton is given in the ".xta" Uppaal input format [9]. The translation gives one timed automaton for each transition and one automaton for the supervisor SU as described in Section 3. The automata for each transition update an array

[8] The last version of Uppaal (3.4.7) is required to read the files produced by Romeo.

[9] see `www.uppaal.com` for further information about Uppaal.

of integers $M[i]$ (which is the number of tokens[10] in place i in the original TPN). For example, the enabledness and firing conditions of a transition t_i such that ${}^\bullet t_i = (1,0,0)$ and $t_i{}^\bullet = (0,0,1)$, are respectively implemented by $M[0] \geq 1$ and $M[2] := M[2] + 1$. Instead of generating one *template automaton* for each transition, we generate as many templates as types of transitions in the original TPN: the type of a transition is the number of input places and output places. For the example of Fig. 3, there are only three types of transitions (one input place to one output place, one to two and two to one) and three templates in the UPPAAL translation. Then one of these templates is instantiated for each transition of the TPN we started with. An example of a UPPAAL template for transitions having one input place and one output place is given in Fig. 4; integers `B1` and `F1` give respectively the index of the unique input place of the transition, and the index of the output place. The timing constraints of the transition are given by `dmin` and `dmax`. We can handle as well transitions with input and output arcs with arbitrary weights (on the examples of Fig. 4 the input and output weights are 1).

In our translation, each transition of the TPN is implemented by a TA with one clock. The synchronized product thus contains as many clocks as the number of transitions of the TPN. At first sight one can think that the translation we have proposed is far too expensive w.r.t. to the number of clocks to be of any use when using a model-checker like UPPAAL: indeed the model-checking of TA is exponential in the number of clocks. Nevertheless we do not need to keep track of all the clocks as many of them are not useful in many states.

5.2 Inactive clocks

When a transition in a TPN is disabled, there is no need to store the value of the clock for this transition: this was already used in the seminal paper of B. Berthomieu and M. Diaz [5]. Accordingly when the TA of a transition is in location $\bar{t}$ (not enabled) we do not need to store the value of the clock: this means that many of the clocks can often be disregarded.

In UPPAAL there is a corresponding notion of *inactive clock*:

Definition 9 (Uppaal inactive clock) *Let $\mathcal{A}$ be a timed automaton. Let x be a clock of $\mathcal{A}$ and ℓ be a location of $\mathcal{A}$. If on all path starting from (ℓ, v) in $S_{\mathcal{A}}$, the clock x is always reset before being tested then the clock x is* inactive

[10] The actual meaning of $M[i]$ is given by a table that is available in the ROMEO tool via the "Translate/Indices $\Longrightarrow$ Place/Transition" menu; the table gives the name of the place represented by $M[i]$ as well as the corresponding information for transitions.

in location ℓ. A clock is active *if it is not inactive.*

A consequence of the notion of inactive clocks in UPPAAL is that at location ℓ the DBM that represents the constraints on the clocks will only contain the active clocks. The next proposition (which is easy to prove on the timed automaton of a transition) states that our translation is *effective* w.r.t. active clocks reduction *i.e.* that when a TA of a transition is not in state t (enabled) the corresponding clock is considered inactive by UPPAAL.

Proposition 1 *Let $\mathcal{A}_i$ be the timed automaton associated with transition t_i of a TPN $\mathcal{T}$ (see Fig. 1, page 10). The clock x_i of $\mathcal{A}_i$ is inactive in locations Firing and $\bar{t}$.*

The current version of UPPAAL (3.4.7) computes active clocks syntactically for each automaton of the product. When the product automaton is computed "on the fly" (for verification purposes), the set of active clocks for a product location is simply the union of the set of active clocks of each component. Again without difficulty we obtain the following theorem:

Theorem 3 *Let $\mathcal{T}$ be a TPN and $\Delta(\mathcal{T})$ the equivalent product of timed automata (see section 3). Let M be a (reachable) marking of $S_{\mathcal{T}}$ and ℓ the equivalent*[11] *location in $S_{\Delta(\mathcal{T})}$. The number of active clocks in ℓ is equal to the number of enabled transitions in the marking M.*

Thanks to this theorem and to the *active clocks reduction* feature of UPPAAL the model-checking of TCTL properties on the network of timed automata given by our translation can be efficient. Of course there are still examples with a huge number of transitions, all enabled at any time that we will not be able to analyze, but those examples cannot be handled by any existing tool for TPN.

In the next subsection we apply our translation to some recent and non trivial examples of TPNs that can be found in [15].

5.3 Tools for analyzing TPNs

One feature of ROMEO is now to export a TPN to UPPAAL or KRONOS but it was originally developed to analyze directly TPNs and has many built-in capabilities: we refer to ROMEO STD for the tool ROMEO with these capabilities [15]. TINA [4] is another state-of-the-art tool to analyze TPNs with

[11] See Definition 6.

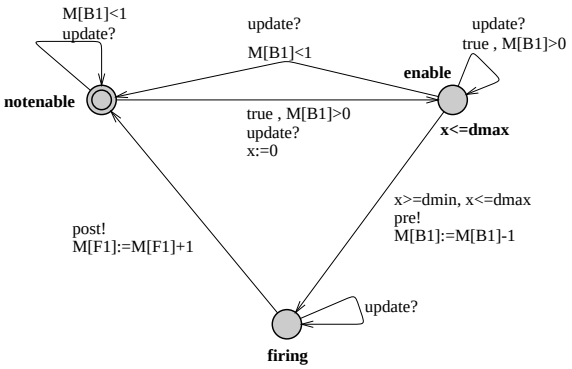


Fig. 4. A UPPAAL template (from "Export to UPPAAL" feature of ROMEO)

some more capabilities than ROMEO STD: it allows to produce an Atomic State Class Graph (ASCG) on which CTL* properties can be checked. Using ROMEO STD or TINA is a matter of taste as both tools give similar results on TPNs.

Table 5.3 gives a comparison in terms of the classes of property (LTL, CTL, TCTL, Liveness) the tools can handle. The columns UPPAAL and KRONOS in ROMEO give the combined capabilities obtained when using our structural translation and the corresponding (timed) model-checker.

Regarding time performance ROMEO STD and TINA give almost the same results. Moreover with ROMEO STD and TINA, model-checking LTL or CTL properties will usually be faster than using ROMEO +UPPAAL: those tools implement efficient algorithms to produce the (A)SCG needed to perform LTL or CTL model-checking. On one hand it is to be noticed that both ROMEO STD and TINA need 1) to produce a file containing the (A)SCG; and then 2) to run a model-checker on the obtained graph to check for the (LTL, CTL or CTL*) property. This can be prohibitive on very large examples as the ones we use in Table 7.

On the other hand neither ROMEO STD nor TINA are able to check quantitative properties such as quantitative liveness (like property of equation (6) below) and TCTL which in general cannot be encoded with an observer (when this possible we can translate such a quantitative property into a problem of marking reachability).

5.4 Checking a liveness quantitative property on $\mathcal{T}_g$.

Let us consider the TPN $\mathcal{T}_g$ of Fig. 6. The *response* (liveness) property,

$$\forall\Box\Big((M[1] > 0 \wedge M[3] > 0 \wedge T_1.x > 3) \Longrightarrow \quad \forall\Diamond(M[2] > 0 \wedge M[4] > 0) \quad (6)$$

<table>
<tr><td rowspan="3"></td><td rowspan="3">TINA</td><td colspan="3">ROMEO</td></tr>
<tr><td rowspan="2"></td><td colspan="2">ROMEO (TPN to TA)</td></tr>
<tr><td>ROMEO STD</td><td>UPPAAL</td><td>KRONOS</td></tr>
<tr><td>Marking Reachability</td><td>Compute marking graph</td><td>On the fly computation (zone-based method)</td><td rowspan="4">UPPAAL-TCTL [c]</td><td rowspan="6">TCTL</td></tr>
<tr><td>LTL</td><td>SCG [a] + MC [b]</td><td>SCG [a] + MC [b]</td></tr>
<tr><td>CTL</td><td>(CTL*) ASCG [a] + MC [b]</td><td rowspan="3">–</td></tr>
<tr><td>Q-liveness [d]</td><td rowspan="2">-</td><td>UPPAAL-Liveness [c]</td></tr>
<tr><td>TCTL</td><td>UPPAAL-TCTL [c]</td></tr>
</table>

[a] SCG = Computation of the State Class Graph ; ASCG = of the atomic SCG.
[b] MC = requires the use of a Model-Checker on the SCG.
[c] UPPAAL implements a subset of TCTL and a special type of liveness defined by formulas of the form $\forall\Box(\varphi \Longrightarrow \forall\Diamond\Psi)$.
[d] Include reponse properties like $\forall\Box(\varphi \Longrightarrow \forall\Diamond\Psi)$ where φ or Ψ can contain clock constraints.

Fig. 5. What can we do with the different tools and approaches?

where $M[i]$ is the marking of the place P_i, cannot be checked with TINA (nor with ROMEO STD) and can easily be checked with our method using the translation and UPPAAL. This property means that if we do not fire T_1 before 3 t.u. then it is unavoidable that at some point in the future there is a marking with a token in P_2 and in P_4. In UPPAAL we can use the response property template `P --> Q` which corresponds to $\forall\Box(P \Longrightarrow \forall\Diamond Q)$. Using our TPN-TCTL translation we obtain:

```
(SU.0 and  M[1]>0 and M[3]>0 and T_1.x>3) -->
                                  (SU.0 and M[2]>0 and M[4]>0)
```

To our knowledge, the translation of TPNs to TA implemented in ROMEO is currently the only existing method allowing the checking of these properties (quantitative liveness and the UPPAAL subset of TCTL) on TPNs.

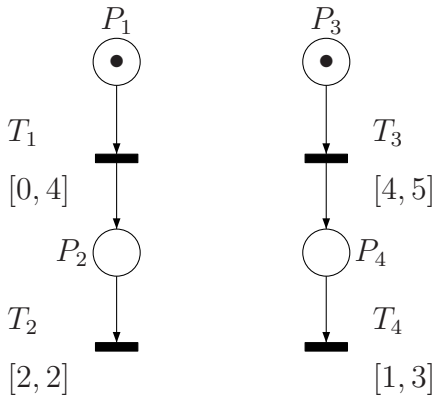


Fig. 6. The TPN $\mathcal{T}_g$

5.5 Experimental results

As a preamble we just point out that our translation is syntactic and the time to translate a TPN into an equivalent product of TA is negligible. This is in contrast with the method used in TINA and ROMEO STD where the whole state space has to be computed in order to build some graph (usually very large) and later on, a model-checker has to be used to check the property on the graph. In Table 7 the time column for TINA refers to the time needed to generate the SCG whereas in the time column for UPPAAL we give the overall time to check a property. The tests have been performed on a Pentium-PC 2GHz with 1GB of RAM running LINUX and with TINA 2.7.2 and UPPAAL 3.4.7 and ROMEO 2.5.0. Hereafter we give the results on three types of examples: classical cyclic or periodic synchronized tasks, producers/consumers and large real-time examples. For all these examples we check a safety property that is true so that UPPAAL must explore the whole state space of the TPN; we can then compare the computation time and memory requirements for UPPAAL and TINA.

5.6 Cyclic and periodic tasks.

This set of examples we call *Real-time classical* examples consists of cyclic and periodic synchronized tasks (like the alternate bit protocol modeled in [5]). On these examples we check the *k-boundedness* property of the TPN and also the property P: $\forall\Box(SU.0 \Longrightarrow M[1] \leq 10)$. k-boundedness is checked by setting the maximum value [12] of the array M to k: in case the bound is hit for one transition, the module `verifyta` of UPPAAL will issue a warning "State discarded" indicating which $M[i]$ was above the bound k. If no warning is issued, all the places are bounded by k. We check property P to force UPPAAL to explore the whole state space of the system. In the sequel we will check for

[12] Thus we check that for all place i, $M[i] \leq k$.

20-boundedness of the TPNs. If the result is "Property P is satisfied" and no warning "State discarded" has been issued we are sure that 1) the TPN is bounded and 2) UPPAAL has scanned the whole state space of the TPN. The results of the use of UPPAAL to check these properties are given in Table 7. All the examples of this category are bounded and satisfy property P.

Except two cases (`oex5,t3`) UPPAAL uses less memory than TINA whereas TINA is faster. This can be due to initial allocation of memory. Also as those case studies are rather small the measured time may not be meaningful. As we will see in the section *Large examples* below, it turns out that memory is the limiting factor when we want to model-check large examples.

5.7 Producers/Consumers examples.

We also experimented on i-Producers/j-Consumers examples (`PiCj`) where the degree of concurrency is usually very high. The results are given in Table 7. The time required to check a property with UPPAAL depends on the number of clocks simultaneously enabled in the TPN. Thus the producers-consumers `P6C7` for which there are always at least 10 transitions simultaneously enabled is a particularly unfavourable case. Of course, it is difficult to generalize from these results since they are highly dependent on the TPN and on the property that we check on it. However, they give an idea of the time and space required for the verification and also shows that our method can be used on non trivial examples. All these examples are bounded and P is satisfied. Again UPPAAL uses far less memory than TINA.

5.8 Large examples.

These examples are very large and with TINA we can only compute the SCG for the first one. For the larger ones, TINA uses two much memory and after a while swapping becomes predominant. For the example `Gros1`, after 10 hours the TINA process had only used 300 seconds of CPU time and was using almost all the virtual memory. No output was produced. The same happened for the examples `Gros2, Gros3`. On these examples, UPPAAL was able to check boundedness and the property P even if the last example (`Gros3`) required ten hours of computation: the maximum amount of memory needed is 150MB and no swapping occurred. As it is generally admitted in the verification community, memory is the limiting factor because you cannot afford many Gigabytes of memory whereas you can wait for a couple of hours. In this way our method seems to be helpful in many cases and worth using.

	File	#places	#trans.	ROMEO +UPPAAL [a]		TINA [b]	
				Time (s)	Mem. (MB)	Time (s)	Mem. (MB)
Real Time Classical Examples	oex1	12	12	0.04	1.3	0.02	1.3
	oex5 [e]	29	23	0.32	3.7	0.00	1.3
	oex7	22	20	1.0	8.9	3.66	162.5
	oex8	31	21	9.50	10	3.53	160
	t001	44	39	5.92	10.5	0.13	13.3
	t3 [e]	56	50	1.18	5	0.0	1.3
Producers Consumers	P3C5	14	13	0.47	5	0.13	7
	P4C5	15	14	1.41	7.5	0.27	20
	P5C5	16	15	1.22	7.5	0.24	16
	P6C7	21	20	26.33	50.8	3.8	111
	P10C10	32	31	1.86	15	0.60	18
Large examples	Gros0	80	72	416	74	20	458
	Gros1	89	80	780	52	N/A [c]	≥ 1300 [d]
	Gros2	116	105	5434	88	N/A [c]	≥ 1300 [d]
	Gros3	143	130	36300	150	N/A [c]	≥ 1300 [d]

[a] We run `verifyta -q -H 65536,65536 -S 1 -s file ''property P''` which is the default.
[b] We run `tina -W -s 0 file` which is the default and generates the SCG.
[c] As the memory consumption is very high, swapping becomes dominant and after 300 sec. when the 1GB RAM is full; the process was killed after 36000 sec. and within this time only 800 sec. were used by TINA and the rest was spent on I/O by the swap process.
[d] This is virtual memory usage when the process was killed.
[e] In this case UPPAAL uses more memory than TINA: this may be due to memory allocation at the beginning of the process.

Fig. 7. Experimental Results: checking a safety property (that is true)

6 Conclusion

In this paper, we have presented a structural translation from TPNs to TA. Any TPN $\mathcal{T}$ and its associated TA $\Delta(\mathcal{T})$ are timed bisimilar.

Such a translation has many theoretical implications. Most of the positive

theoretical results on TA carry over to TPNs. The class of TPNs can be extended by allowing strict constraints (open, half-open or closed intervals) to specify the firing dates of the transitions; for this extended class, the following results follow from our translation and from Theorem 1:

- TCTL model checking is decidable for bounded TPNs. Moreover efficient algorithms used in UPPAAL [22] and KRONOS [30] are exact for the class of TA obtained with our translation (see recent results [10] by P. Bouyer);
- it is decidable whether a TA is non-zeno or not [16] and thus our result provides a way to decide non-zenoness for bounded TPNs;
- lastly, as our translation is structural, it is possible to use a model-checker to find sufficient conditions of unboundedness of the TPN.

These results enable us to use algorithms and tools developed for TA to check quantitative properties on TPNs. For instance, it is possible to check real-time properties expressed in TCTL on bounded TPNs. The tool ROMEO [15] that has been developed for the analysis of TPN (state space computation and "on the fly" model-checking of reachability properties) implements our translation of a TPN into the equivalent TA in UPPAAL input format.

Our approach turns out to be a good alternative to existing methods for verifying TPNs:

- with our translation and UPPAAL we were able to check safety properties on very large TPNs that cannot be handled by other existing tools;
- we also extend the class of properties that can be checked on TPNs to real-time quantitative properties.

Note also that using our translation, we can take advantage of all the features of a tool like UPPAAL: looking for counter examples is usually much faster than checking a safety property. Moreover if a safety property is false, we will obtain a counter example even for unbounded TPNs (if we use breadth-first search).

References

[1] P. A. Abdulla and A. Nylén. Timed Petri nets and BQOs. In *ICATPN'01*, volume 2075 of *LNCS*, pages 53–72. Springer-Verlag, June 2001.

[2] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science B*, 126:183–235, 1994.

[3] T. Aura and J. Lilius. A causal semantics for time Petri nets. *Theoretical Computer Science*, 243(1–2):409–447, 2000.

[4] B. Berthomieu, P.-O. Ribet and F. Vernadat. The tool TINA – Construction of Abstract State Spaces for Petri Nets and Time Petri Nets. *International Journal of Production Research*, 4(12), July 2004.

[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, March 1991.

[6] B. Berthomieu and M. Menasche. An enumerative approach for analyzing time Petri nets. In *Information Processing*, volume 9 of *IFIP congress series*, pages 41–46. Elsevier Science Publishers, 1983.

[7] B. Berthomieu and F. Vernadat. State class constructions for branching analysis of time Petri nets. In *TACAS'2003*, volume 2619 of *LNCS*, pages 442–457, 2003.

[8] S. Bornot, J. Sifakis, and S. Tripakis. Modeling urgency in timed systems. *COMPOS'97, Malente, Germany*, volume 1536 of *LNCS*, pages 103–129, 1998.

[9] L. A. Cortés, P. Eles, Z. Peng Verification of Embedded Systems using a Petri Net based Representation, 13th International Symposium on System Synthesis (ISSS'2000), Madrid, Spain, pages 149-155, 2000.

[10] P. Bouyer. Untameable timed automata! In *STACS'2003*, volume 2607 of *LNCS*, pages 620–631, 2003.

[11] P. Bouyer, C. Dufourd, E. Fleury, and A. Petit. Are timed automata updatable? In *CAV'2000*, volume 1855 of *LNCS*, pages 464–479, 2000.

[12] Franck Cassez and Olivier H. Roux. Structural Translation of Time Petri Nets into Timed Automata. In Michael Huth, editor, *Workshop on Automated Verification of Critical Systems (AVoCS'04)*, Electronic Notes in Computer Science. Elsevier, August 2004.

[13] D. de Frutos Escrig, V. Valero Ruiz, and O. Marroquín Alonso. Decidability of properties of timed-arc Petri nets. In *ICATPN'00*, Aarhus, Denmark, volume 1825 of *LNCS*, pages 187–206, june 2000.

[14] M. Diaz and P. Senac. Time stream Petri nets: a model for timed multimedia information. In *ATPN'94*, volume 815 of *LNCS*, pages 219–238, 1994.

[15] G. Gardey, D. Lime, and O. (H.) Roux. ROMÉO: A tool for Time Petri Nets Analysis. The 17th International Conference on Computer Aided Verification (CAV'05), Edinburgh, Scotland, volume 3576 of *LNCS*, pages 418–423, july 2005. tool can be freely downloaded from `http://romeo.rts-software.org/`.

[16] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.

[17] W. Khansa, J.P. Denat, and S. Collart-Dutilleul. P-Time Petri Nets for manufacturing systems. In *WODES'96*, Scotland, pages 94–102, 1996.

[18] F. Laroussinie and K. G. Larsen. CMC: A tool for compositional model-checking of real-time systems. In *FORTE-PSTV'98*, pages 439–456. Kluwer Academic Publishers, 1998.

[19] J. Lilius. Efficient state space search for time Petri nets. *Electronic Notes in Theoretical Computer Science*, volume 18, 1999.

[20] D. Lime and O. H. Roux. State class timed automaton of a time Petri net. In *PNPM'03*. IEEE Computer Society, September 2003.

[21] P. M. Merlin. *A study of the recoverability of computing systems*. PhD thesis, University of California, Irvine, CA, 1974.

[22] P. Pettersson and K. G. Larsen. UPPAAL2k. *Bulletin of the European Association for Theoretical Computer Science*, 70:40–44, February 2000.

[23] M. Pezzé and M. Young. Time Petri Nets: A Primer Introduction. Tutorial presented at the Multi-Workshop on Formal Methods in Performance Evaluation and Applications, Zaragoza, Spain, September 1999.

[24] L. Popova. On time Petri nets. *Journal Information Processing and Cybernetics, EIK*, 27(4):227–244, 1991.

[25] C. Ramchandani. *Analysis of asynchronous concurrent systems by timed Petri nets*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1974.

[26] A. T. Sava. *Sur la synthèse de la commande des systèmes à évènements discrets temporisés*. PhD thesis, INPG, Grenoble, France, November 2001.

[27] J. Sifakis. Performance Evaluation of Systems using Nets. In *Net Theory and Applications, Advanced Course on General Net Theory of Processes and Systems, Hamburg*, volume 84 of *LNCS*, pages 307–319, 1980.

[28] J. Sifakis and S. Yovine. Compositional specification of timed systems. In *STACS'96*, volume 1046 of *LNCS*, pages 347–359, 1996.

[29] S. Tripakis. Timed diagnostics for reachability properties. In W.R. Cleaveland, editor, *(TACAS'99)*, volume 1579 of *LNCS*, pages 59–73, 1999.

[30] S. Yovine. Kronos: A Verification Tool for real-Time Systems. *Journal of Software Tools for Technology Transfer*, 1(1/2):123–133, October 1997.