Formal verification of real-time systems with preemptive scheduling

Didier Lime and Olivier (H.) Roux

IRCCyN (Institut de Recherche en Communication et Cybernétique de Nantes)
1, rue de la Noë B.P. 92101
44321 NANTES cedex 3 (France)
{Didier.Lime | Olivier-h.Roux}@irccyn.ec-nantes.fr

Abstract. In this paper, we propose a method for the verification of timed properties for real-time systems featuring a preemptive scheduling policy: the system, modeled as a scheduling time Petri net, is first translated into a linear hybrid automaton to which it is time-bisimilar. Timed properties can then be verified using HyTech. The efficiency of this approach leans on two major points: first, the translation features a minimization of the number of variables (clocks) of the resulting automaton, which is a critical parameter for the efficiency of the ensuing verification. Second, the translation is performed by an over-approximating algorithm, which is based on Difference Bound Matrix and therefore efficient, that nonetheless produces a time-bisimilar automaton despite the over-approximation. The proposed modeling and verification method are generic enough to account for many scheduling policies. In this paper, we specifically show how to deal with Fixed Priority and Earliest Deadline First policies, with the possibility of using Round-Robin for tasks with the same priority. We have implemented the method and give some experimental results illustrating its efficiency.

1 Introduction

Hard real-time systems are both complex and critical. Therefore thorough verification processes of such systems must be performed, including behavior and timing correctness. These systems are usually designed as a set of several tasks interacting and sharing one ore more processors. Hence, in a system S, tasks must be scheduled on the processors in such a way that they respect some properties P_i imposed by the controlled process. This is usually achieved using either an offline or an online approach. In the offline approach, a pre-runtime schedule is built up so that S satisfies P_i. In the online approach, the schedule of the tasks is computed at runtime according to a scheduling policy based on priorities (e.g. Rate Monotonic, Earliest Deadline First). Among the P_i properties S must verify, the first one is the schedulability, i.e. "*each task meets its deadline*". In this paper, we shall consider the online approach.

1.1 Analytical online scheduling analysis

The online scheduling analysis is a much studied topic and many analytical results have been proposed, mostly concerning the schedulability of task sets since the seminal work of Liu and Layland in 1973 (Liu and Layland 1973). Results on low-cost exact analysis of sets of independent and periodic tasks with fixed execution times are presented in (Tindell 1994, Palencia and Harbour 1998, Hladik and Déplanche 2003) for instance. Extensions have been proposed to take into account interactions between tasks and variable execution time, see in particular (Palencia and Harbour 1999, Harbour *et al.* 1991). The authors give upper bounds of response times and thus only sufficient conditions. This leads to an inherent pessimism, which potentially grows with the complexity of the system considered. This motivates the use of formal verification methods using such models as timed automata (TA) (Alur and Dill 1994, Henzinger *et al.* 1994) and timed Petri nets (TPN) (Merlin 1974, Berthomieu and Diaz 1991), among others.

1.2 Formal models for online scheduling

Timed models such as timed automata or time Petri nets are usually not expressive enough to model and verify many classical features of real-time systems: in these models, time elapses at the same speed for all components of the system, which allows the modeling of non-preemptive scheduling policies of tasks, each being executed on a different processor. However, they cannot represent preemptive scheduling policies where the execution of a task can be suspended at some point and later resumed at the same point.

Automata and scheduling. Hybrid automata extend finite automata with continuous variables whose dynamic evolution is specified in each location by a differential equation. Linear hybrid automata (LHA) (Alur *et al.* 1995) are a subclass of hybrid automata. In this model, the differential equation governing the evolution of the variables X is written $A\dot{X} \leq B$, with A being a real matrix and B a real vector. For this subclass, symbolic verification (semi-)algorithms have been developed (Alur *et al.* 1996) and implemented in the HyTech tool (Henzinger *et al.* 1997).

Stopwatch automata (SWA) (Cassez and Larsen 2000) can be defined as timed automata for which clocks can be stopped and later resumed with the same value. These clocks are then more appropriately called stopwatches. This is thus a syntactical subclass of LHA which is of particular interest when modeling real-time systems. Cassez and Larsen show however in (Cassez and Larsen 2000) that stopwatch automata (with unobservable delays) are as expressive as LHA in the sense that for all timed language accepted by a LHA, there exists a stopwatch automaton which accepts the same language. The reachability problem being undecidable for LHA, this shows that it is also undecidable for stopwatch automata. The authors then propose an over-approximation of the

reachable state-space using Difference Bound Matrix (DBM) (Berthomieu and Menasche 1983, Dill 1989).

In (McManis and Varaiya 1994), McManis and Varaiya propose an extension of timed automata, called suspension automata, where continuous variables progress similarly as exposed in (Cassez and Larsen 2000). They prove that, for this model, reachability is undecidable and fall back to a decidable case, similar to that of timed automata, by considering that the suspension durations are fixed and integral. When a stopwatch is stopped and resumed, the duration of the suspension is subtracted from its value. In this approach, precedence relations inducing preemptions have to be explicitly encoded by locations of the automaton, which may restrict the expressivity and the ease of use of the model. In addition, fixed execution times cannot account for a number of well-known cases in which the early termination of a task leads to a longer response time for another task.

The approach of McManis and Varaiya is further developed by Fersman, Mokrushin, Petterson and Yi (Fersman *et al.* 2002, Fersman *et al.* 2003). Using so-called task automata, modeling task arrivals, the authors propose an analysis of the schedulability of the system as a reachability problem in a subtraction automaton modeling the scheduler. In these paper, the authors have found a way to avoid stopwatches and get around the undecidability problems. However, in (Fersman *et al.* 2002) task execution times are given as intervals but tasks are independent. In (Fersman *et al.* 2003, Fersman and Yi 2004), the authors introduce shared resources and semaphores in their analysis but the execution times of tasks are fixed and it is easy to show that, in this setting, reducing the execution time of a task may decrease the overall timing performances for the application. Finally, in (Fersman *et al.* 2006) the authors consider the problem of fixed priority scheduling policy when the task execution times are given as intervals but, as the problem is undecidable, they propose an over-approximation technique.

Finally, an interesting approach, proposed by Altisen *et al.* (Altisen *et al.* 1999, Altisen *et al.* 2000, Altisen *et al.* 2002) uses the paradigm of controller synthesis, modeling the scheduler as a controller of the system. The idea is then to obtain, by construction, the schedulability of the system modeled using a discrete time semantics (according to the authors, the methodology can be adapted to continuous timed model modulo some technical problems related to time density), by restricting the guards of controllable events. These restrictions are obtained by adding control invariants modeling the schedulability constraints and the scheduling policy. This method may however be hard to use in practice, as the authors acknowledge that the search for control invariants may be difficult.

Time Petri nets and scheduling. A number of authors propose extensions of time Petri nets to account for the suspension and resumption of actions.

(Okawa and Yoneda 1996) propose an approach with time Petri nets where groups of transitions are defined together with rates (speeds) of execution. Transition groups correspond to transitions that model concurrent activities and that

can become simultaneously firable. In this case, each rate is then divided by the sum of transition execution rates.

Roux and Déplanche (Roux and Déplanche 2002) propose an extension for time Petri nets (called *Scheduling*-TPN) that allows to take into account the way the real-time tasks of an application distributed over different processors are scheduled. They propose the computation of an over-approximation of the state space based on the classical state-class graph of (Berthomieu and Diaz 1991) and DBM.

The same approach is developed in (Bucci *et al.* 2004, Bucci *et al.* 2003), with preemptive time Petri nets (*Preemptive*-TPNs). In (Bucci *et al.* 2004), the authors also propose an interesting method allowing the timed analysis of the net: given an untimed transition sequence from the over-approximated state-class graph, they retrieve the possible durations between the firings of the transitions in the sequence, as the solutions of a linear programming problem. They can thus verify if the sequence is actually possible in the net or if it was added by the over-approximation.

These last two models are subclasses of inhibitor hyperarcs time Petri nets (IHTPN) (Roux and Lime 2004), which, since they are more general, are not as well-suited for modeling real-time systems. For these three models, reachability is undecidable even when the net is bounded (*i.e.*, there exists a constant k such that for all reachable states, the number of tokens in any place is less than k) (Berthomieu *et al.* 2007). Reachability and related results are then obtained by computing the state-space if it is computable.

1.3 Issues of formal verification of real-time systems

State-space computation. With a discrete-time semantics, the state-space is generally finite but the analysis is hindered by the well-known state explosion problem, even with the use of acceleration techniques available in tools such as Fast (Bardin *et al.* 2003).

For dense-time models, the state-space is generally infinite, because of the real-valued clocks, so, one needs to group some states together, in order to obtain a finite number of these groups, which is hopefully computable. These groups of states are, classically, *regions* and *zones* for timed automata or *state classes* for time Petri nets. If the model does not feature any stopwatch, then the states contained in those groups may be described by linear inequations of a particular type which may be encoded into a so-called Difference Bound Matrix (DBM) (Dill 1989, Berthomieu and Menasche 1983). DBM allow fast manipulation and generation (*i.e.* polynomial complexity). When the model has more than one evolution rate for its continuous variables (e.g. when featuring stopwatches), inequations describing the group of states are more complex and can non longer be described as a DBM. A general polyhedron representation is needed, which involves a much more complex manipulation and generation cost (*i.e.* exponential complexity). As a consequence, an idea to speed up the state-space computation is to expand the general polyhedra into DBM. This is clearly an over-approximation.

Number of clocks. The number of clocks/variables is a critical concern with the verification of formal models. Generating and handling polyhedra in the general case are operations that have a complexity that is exponential in the number of variables of the polyhedron. In the case of hybrid systems such as SWA, these variables are the stopwatches. With the increase of the number of stopwatches, the analysis quickly becomes intractable with a tool for linear hybrid automata such as HyTech (Henzinger *et al.* 1997). Algorithms have been developed for timed automata, such as (Daws and Yovine 1996), to reduce the number of clocks. To our knowledge, there are no such algorithm for hybrid automata. Moreover, these algorithms cannot be applied to products of automata, which are (heavily) used to properly model real-time systems with SWA.

1.4 Our contribution

In this paper, we consider the *Scheduling*-TPN model for its ability to conveniently model concurrency and for the fact that the upper bound of countable resources such as semaphores and queues do not need to be given *a priori* in a Petri net-based model. Moreover, the adequacy of *Scheduling*-TPN to the modeling of classical services provided by real-time executives (shared resource access protocol, task activation, synchronization and messaging) and the modeling of classical components of distributed embedded systems such as CAN buses have been exposed in (Lime and Roux 2003). Similarly to formalisms such as ACSR (Brémond-Grégoire *et al.* 1993), we can then model and analyze arbitrarily complex task behaviors.

For *Scheduling*-TPN, we tackle the problem of the state-space explosion by a two-stage analysis. First, we pre-compute the state space of the *Scheduling*-TPN as a linear hybrid automaton. This first step is performed by a fast DBM-based algorithm. Although this algorithm is over-approximating, the produced linear hybrid automaton is proved to be time-bisimilar to the initial *Scheduling*-TPN *i.e.* the additional locations generated by the approximation are actually not reachable. As a consequence, the cost of the translation is fairly low. The second step consists of an exact analysis of that LHA with the HyTech model-checker. For this second step to be efficient, the number of variables (clocks) must be kept as low as possible. To this effect, the translation algorithm offers a number of reduction mechanisms and thus produces a LHA that has, in general, a fairly lower number of variables than what is required for a direct modeling as a product of linear hybrid automata.

This paper extends results presented in both (Lime and Roux 2004) and (Lime and Roux 2006*b*). These previous papers focused on the fixed priority scheduling policy. Here we have clearly separated the usual mechanics of time Petri nets and the scheduling information and we have extended both the model and the associated state-space computation algorithms to take Earliest Deadline First (EDF) and Round-Robin scheduling policies into account.

In order to get a proper view of the application area of our method, let us consider a real-time system with a preemptive scheduling policy to analyze. Let us restrict ourselves to the schedulability verification problem:

– for independent periodic tasks, one should definitely go for the low cost exact analysis of Liu and Layland (Liu and Layland 1973) for EDF and Tindell (Tindell 1994) for fixed priority and subsequent works along these lines ;
– in the presence of interactions between tasks, it becomes interesting to use formal methods: if the execution times of tasks are exactly known then the method by Fersman *et al.* (Fersman *et al.* 2006) is good choice ;
– for interacting tasks with variable execution times, the modeling requires stopwatches in a dense-time setting and the verification may be performed using hybrid models such as LHA (Alur *et al.* 1995) or Petri nets with stopwatches for model-checking (Bucci *et al.* 2004, Roux and Lime 2004). The method described in this paper follows this line of works and proposes a formalism that is conveniently taking into account real-time systems features, including major scheduling policies like EDF or fixed priority, and translating it into a model optimized in terms of analysis efficiency.

Outline of the paper

The rest of the paper is organized as follows: section 2 presents the *Scheduling*-TPN model and its instances for Fixed Priority, Earliest Deadline First and Round-Robin schedulers. Section 3 adapts the classical state-class graph method of Berthomieu *et al.* to that model. Section 4 describes the translation into a linear hybrid automaton and proves the correctness of the translation by proving a time-bisimulation relation. Finally, we apply our results on an example in section 5.

2 A formal model for real-time systems

Computer control systems are essentially discrete. Therefore, any such system can be modeled by a discrete-event system with a discrete time assumption. For instance, for one processor (CPU), the discrete time step can be chosen as the cycle time of the CPU or the tick time of the scheduler. This is however highly inefficient and actually unusable because the number of states to analyze is much too large: this is an instance of the so-called "state-space explosion problem".

Consequently, we choose a higher level model in which:

– the cycle-wise execution of a task is modeled by the continuous evolution of a variable representing the time during which the task has been executed;
– the preemption of a task is modeled by stopping the evolution of that variable (making its derivative equal to 0);
– the cyclic preemptions induced by the round-robin scheduling policy of n tasks sharing the same processor are modeled by an evolution of the corresponding variables with the derivatives $\frac{1}{n}$.

For this purpose, we introduce a powerful new model for real-time systems. It consists of two layers: the formal layer and the scheduling layer.

The formal layer is based on time Petri nets and allows the modeling of task structure, synchronizations, communications and timings. The scheduling layer gives the resource requirements of tasks and scheduling policies associated to them, as well as all the information needed by the scheduler to operate (priority, deadlines, ...).

Given a state of the system, the scheduling layer output is the resource allocation, which is then used by the formal layer to update the state of the system. The new state is then sent back to the scheduling layer.

Thanks to the above modeling choices, time elapsing can be abstracted within the formal layer and the new state that is passed back to the scheduling layer corresponds to what is obtained after the next discrete event (firing of a transition). Therefore, the allocation of resources is constant between two discrete events.

2.1 The Formal Layer

Notations. We denote A^X the set of mappings from X to A. If X is finite and $|X| = n$, an element of A^X is also a vector in A^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* $\nu \in A^X, d \in A$, $\nu + d$ denotes the vector such that $(\nu + d)(x) = \nu(x) + d$ and for $X' \subseteq X$, $\nu[X' \mapsto 0]$ denotes the valuation ν' with $\nu'(x) = 0$ for $x \in X'$ and $\nu'(x) = \nu(x)$ otherwise.

Syntax. The following definition formally defines *Scheduling*-TPNs.

Definition 1 (*Scheduling*-TPN). *A scheduling time Petri net (Scheduling-TPN) is a 8-tuple* $\mathcal{T} = (P, T, {}^\bullet(.), (.)^\bullet, \alpha, \beta, M_0, \mathsf{NewFlow})$*, where*

- $P = \{p_1, p_2, \ldots, p_m\}$ *is a finite non-empty set of* places*,*
- $T = \{t_1, t_2, \ldots, t_n\}$ *is a finite non-empty set of* transitions *($T \cap P = \emptyset$),*
- ${}^\bullet(.) \in (\mathbb{N}^P)^T$ *is the* backward incidence function*,*
- $(.)^\bullet \in (\mathbb{N}^P)^T$ *is the* forward incidence function*,*
- $M_0 \in \mathbb{N}^P$ *is the* initial marking *of the net,*
- $\alpha \in (\mathbb{Q}^+)^T$ *and* $\beta \in (\mathbb{Q}^+ \cup \{\infty\})^T$ *are functions giving respectively the* earliest *and* latest *firing times of transitions ($\alpha \leq \beta$),*
- $\mathsf{NewFlow} \in \mathbb{R}^{+^{T^{\mathbb{N}^P \times \mathbb{R}^T}}}$ *is the* activity function*.*

The function NewFlow is the distinctive trait between *TPNs* and *Scheduling*-TPNs. It models the allocation of resources to transitions by the scheduler by giving to each transition the rate at which time will elapse for it. For instance, the rate of progress of a transition modeling a preempted task will be 0, while the rate of a task running alone on its processor will be 1. An in-depth explanation will be given in section 2.3.

Semantics. A *marking* is a function associating to each place of the net the number of tokens it contains.

As usual, a transition t is said to be *enabled* by the marking M if it has "enough" tokens in its input places: $M \geq {}^\bullet t$. We denote by $\mathsf{enabled}(()M)$ the set of transitions enabled by the marking M.

A state of the *Scheduling*-TPN is defined as triple (M, ν, Flow), where M is the marking of the net, ν the function that assigns to each transition the time during which it has been enabled and Flow a function giving for each transition t the speed at which $\nu(t)$ increases.

In this paper, in order to decide which transition clock should be reset when firing a transition, we consider the *intermediate semantics* for TPNs, based on (Berthomieu and Diaz 1991, Aura and Lilius 2000), which is the most common one. The key point in the semantics is to define when a transition is *newly enabled* and its clock must be reset.

Let $\uparrow \mathsf{enabled}(t', M, t) \in \mathbb{B}$ be true if t' is *newly enabled* by the firing of transition t from marking M, and false otherwise. The firing of t leads to a new marking $M' = M - {}^\bullet t + t^\bullet$. The fact that a transition t' is newly enabled on the firing of a transition $t \neq t'$ is determined w.r.t. the intermediate marking $M - {}^\bullet t$. When a transition t is fired it is newly enabled regardless of what the intermediate marking is. Formally this gives:

$$\uparrow \mathsf{enabled}(t', M, t) = (t' \in \mathsf{enabled}(M - {}^\bullet t + t^\bullet) \wedge \big(t' \notin \mathsf{enabled}(M - {}^\bullet t) \vee (t = t')\big)$$

Definition 2 (Semantics of a *Scheduling*-TPN). *The semantics of a Scheduling-TPN $\mathcal{T}$ is defined as non deterministic*[1] *timed transition system (TTS) $S_\mathcal{T} = (Q, Q_0, \rightarrow)$ such that:*

- $Q = \mathbb{N}^P \times (\mathbb{R}^+)^T \times (\mathbb{Q}^+)^T$;
- $Q_0 = \{(M_0, \overline{0}, \mathsf{Flow}_0), \mathsf{Flow}_0 \in \mathsf{NewFlow}(M_0, \overline{0})\}$;
- $\rightarrow \in Q \times (T \cup \mathbb{R}) \times Q$ *is the transition relation including continuous transitions and discrete transitions:*
  - *the* continuous *transition relation is defined* $\forall d \in \mathbb{R}^+$ *by:*

$$(M, \nu, \mathsf{Flow}) \xrightarrow{d} (M, \nu', \mathsf{Flow}) \text{ iff } \begin{cases} \forall t_i \in \mathsf{enabled}(M), \nu'(t_i) = \nu(t_i) + \mathsf{Flow}(t_i) * d, \\ \forall t_k \in T, M \geq {}^\bullet t_k \Rightarrow \nu'(t_k) \leq \beta(t_k). \end{cases}$$

  - *the* discrete *transition relation is defined* $\forall t_i \in T$ *by:*

$$(M, \nu, \mathsf{Flow}) \xrightarrow{t_i} (M', \nu', \mathsf{Flow}') \text{ iff } \begin{cases} t_i \in \mathsf{enabled}(M) \wedge \mathsf{Flow}(t_i) \neq 0, \\ \alpha(t_i) \leq \nu(t_i) \leq \beta(t_i), \\ M' = M - {}^\bullet t_i + t_i{}^\bullet, \\ \forall t_k, \nu'(t_k) = \begin{cases} 0 \text{ if } \uparrow \mathsf{enabled}(t_k, M, t_i), \\ \nu(t_k) \text{ otherwise.} \end{cases} \\ \mathsf{Flow}' \in \mathsf{NewFlow}(M', \nu') \end{cases}$$

[1] The non determinism arises no theoretical problem. For a discussion on its practical implications, see 2.3.

A transition t is said to be *active* for a given state (M, ν, Flow) of the net ρ if t is enabled and $\mathsf{Flow}(t) \neq 0$.

In previous papers, the activity of transitions was defined using a function named *Act* giving a subset of the marking of the net (Roux and Déplanche 2002, Lime and Roux 2003, Lime and Roux 2004, Magnin *et al.* 2005). The transitions enabled by this subset were declared *active*. Now, we need a more general definition to take into account the Round-Robin and Earliest Deadline First scheduling policies. We can fall back to the former setting with the following definition: for a state (M, ν, Flow), $\forall t \in \mathsf{enabled}(M), \mathsf{Flow}(t) = 1$ if $t \in \mathsf{enabled}(Act(M))$ and $\mathsf{Flow}(t) = 0$ otherwise.

Note the following important property of *Scheduling*-TPNs:

Property 1. From a state (M, ν, Flow), Flow does not change by letting time elapse.

The function $\mathsf{NewFlow}(M', \nu')$ is given by the scheduling layer. It is computed after the firing of a discrete transition and gives the set of possible activation choices of the scheduler wrt. the scheduling policy.

2.2 The Scheduling Layer

On top of the formal layer, we define the specifics of our application with respect to the scheduling problem, in terms of tasks, processors,*etc.*

Let *Procs* be the set of processors. We denote by $Sched : Procs \mapsto \{EDF, FP\}$ the function that maps a processor to a scheduling policy, which can be either "Ealiest Deadline First" (EDF) or "Fixed Priority" (FP).

Let *Tasks* be the set of tasks of the system. We assume that there is no task migration and we denote by $\Pi : Tasks \mapsto Procs$ the function that maps a task to its processor.

For tasks τ such that $Sched(\Pi(\tau)) = FP$, the partial function $\omega : Tasks \mapsto \mathbb{N}$ gives the priority of the task on the processor. Similarly, for tasks τ such that $Sched(\Pi(\tau)) = EDF$, the partial function $\delta : Tasks \mapsto \mathbb{N}$ gives the deadline of the task relative to its activation time.

Now we map each place of the net to a task with the function $\gamma : P \mapsto Tasks \cup \{\phi\}$. ϕ is a special element which denotes that the place is not mapped to any real task, for instance because it models a service of the operating system. ϕ can be seen as a special task that is always running.

We assume that for each transition, there is *at most* one place p such that $p \in {}^\bullet t$ and $\gamma(p) \neq \phi$. If $\forall p \in {}^\bullet t, \gamma(p) = \phi$, then t is not bound to any real task and we say that it is *part of* the special task ϕ (denoted by $\gamma(t) = \phi$). Otherwise, for each transition t, we say that t is *part of* the task τ, and we denote it $t \in \tau$ if one of its input places is mapped to τ: $t \in \tau \Leftrightarrow \exists p \in {}^\bullet t$, s.t. $\gamma(p) = \tau$. For convenience, and thanks to the hypothesis above, we denote by $\gamma(t)$ the task s.t. $t \in \tau$.

Each task τ is thus modeled by a subnet of the *Scheduling*-TPN composed of places mapped to τ by γ and of transitions which are part of τ.

We assume that at most one instance of each task is active at a given instant, which is expressed by the restriction that at most one place mapped to τ by γ is marked at a given instant.

Let $B(\tau)$ be the set of transitions which *start* the task τ and similarly, let $E(\tau)$ be the set of transitions which *terminate* τ. These two sets are user-defined as part of the modeling phase.

Example 1 Figure 1 gives an example of the above modeling. It uses design patterns presented in (Lime and Roux 2003). In particular, p_1 and t_1 model the periodic activation of task τ_1. There is only one processor scheduled with EDF, so no priority function ω is defined, only the deadlines of tasks δ. Places p_2 and p_3 are part of task τ_1 and p_4 of task τ_2. So t_2 and t_3 are part of τ_1 and t_4 is part of τ_2. In addition, t_1 begins τ_1 and t_3 ends it; t_2 begins τ_2 and t_4 ends it.

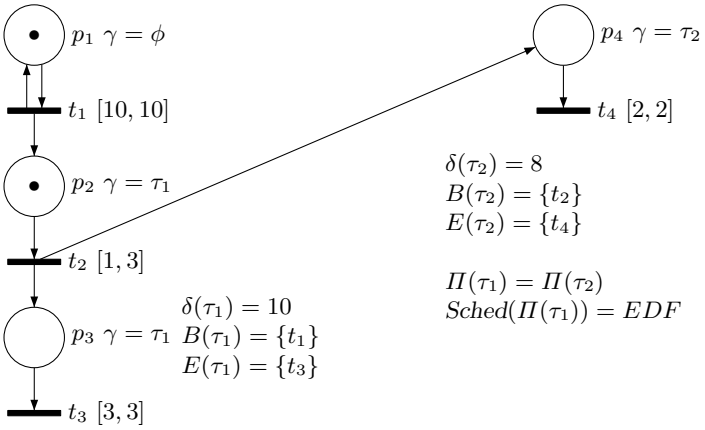

Fig. 1. A *Scheduling*-TPN

Example 2 The net in Figure 2 models two periodic tasks on the same processor with a fixed priority scheduling policy.

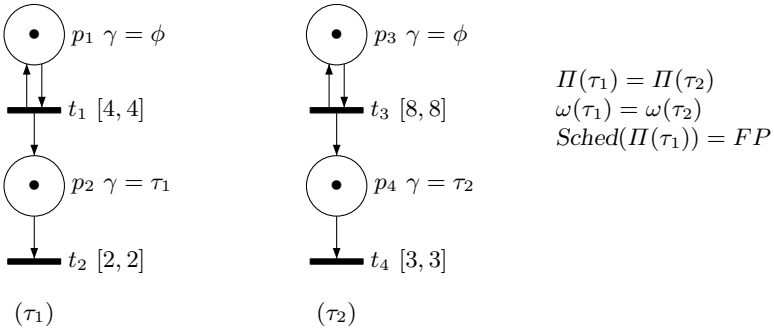

Fig. 2. A system with Fixed Priority scheduling modeled as a *Scheduling*-TPN.

2.3 Computing the activity function

The interface between the formal layer and the scheduling layer is the activity function NewFlow. Its computation (performed after the firing of each discrete transition) models the scheduling policy.

It can model many scheduling policies (including non-deterministic ones), the main constraint being that a rescheduling cannot happen by simply letting time elapse: a discrete event must occur, which is usually the case in practice.

In our setting, exactly one scheduling policy is bound to a given processor. We could model systems with multiple scheduling policies for a given processor by wrapping them up into one higher level policy that decides which policy to use at a given instant.

Given a state of the system[2], NewFlow gives the set of possible activation choices made by the scheduler. Each of these choices is a function that maps a rational number to each enabled transition, representing the speed at which the value of the clock associated to the transition increases. If the scheduler is deterministic, then this set of possible choices is reduced to a unique element.

In the following, we investigate the case of a fixed priority policy, then the case of an Earliest Deadline First policy. In both cases, the scheduler is deterministic so we show how the unique result Flow of the computation of NewFlow is obtained.

Let $s = (M, v, \mathsf{Flow})$ be a state of (the semantics) of the net such that a transition has just been fired (meaning no time has elapsed since). Let t be an enabled transition. If $\gamma(t) = \phi$ then $\mathsf{Flow}(t) = 1$. This models services of the executive. For the following subsections we consider that $\gamma(t) \neq \phi$.

[2] For the sake of simplicity, we restrict NewFlow to states but it could easily be extended to executions (runs), so that the full history of the system can be used by the scheduler.

Fixed Priority. Here we consider the case when $\mathit{Sched}(\Pi(\gamma(t))) = FP$. In this case, if all task priorities on a given processor are different, the situation is quite clear: let $s = (M, v, \mathsf{Flow})$ be a state of (the semantics) of the net. If $\gamma(t)$ has the highest priority $\omega(\gamma(t))$ on its processor then $\mathsf{Flow}(t) = 1$ else $\mathsf{Flow}(t) = 0$. Further steps should be taken if, at some point, some tasks have the same priority on a given processor and this priority happens to be the highest. Before we deal with this problem, let us explain the case of EDF.

Earliest Deadline First. We now consider the case when $\mathit{Sched}(\Pi(\gamma(t))) = EDF$. Let $\mathcal{D} : \mathit{Tasks} \times \mathbb{N}^P \times \mathbb{R}^T \mapsto \mathbb{R}^+$ be the function that gives for each task the time remaining until it reaches its deadline. As a shorthand, for a state $s = (M, \nu, \mathsf{Flow})$ and a task τ, we note $\mathcal{D}(\tau, s) = \mathcal{D}(\tau, M, \nu)$. Then Flow is simply given for a state (M, ν, Flow) by:

$$\mathsf{Flow}(t) = \begin{cases} 1 \text{ if } \mathcal{D}(\gamma(t), M, \nu) = \min_{\tau \in \mathit{Tasks} \text{ s.t. } \Pi(\tau) = \Pi(\gamma(t))} \{\mathcal{D}(\tau, M, \nu)\} \\ 0 \text{ otherwise} \end{cases}$$

Now, the problem is reduced to the computation of the function $\mathcal{D}$. Let τ be a task in *Tasks*. Suppose that each time a transition t_b which begins τ is fired, we dynamically add a clock x_τ to the model, whose initial value is 0. Subsequently, the first time that a transition t_e which ends τ is fired, the clock is removed.

Since a clock is implicitly associated to each enabled transition in time Petri nets, x_τ is actually implemented as an extra transition in the net: for the task τ, let us add an extra transition $\mathcal{D}_\tau$ and an extra place $\mathcal{P}_\tau$ such that:

- $\mathcal{P}_\tau$ is the only input place of $\mathcal{D}_\tau$;
- $\mathcal{D}_\tau$ is set to fire at the same time as the expiration of the deadline of τ: $\alpha(\mathcal{D}_\tau) = \beta(\mathcal{D}_\tau) = \delta(\tau)$;
- $\mathcal{D}_\tau$ is always active, and progressing at rate 1: $\gamma(\mathcal{P}_\tau) = \phi$;
- $\mathcal{D}_\tau$ is enabled when the task τ is started: $\mathcal{P}_\tau$ is an output place for all the transitions in $B(\tau)$ and $M_0(\mathcal{P}_\tau) = 0$;
- $\mathcal{D}_\tau$ is disabled when the task τ ends: $\mathcal{P}_\tau$ is an input place for all the transitions in $E(\tau)$;
- $\mathcal{D}_\tau$ cannot be fired if some transition in $E(\tau)$ is firable.

Now, for each state (M, ν, Flow) between the firings of t_b and t_e we have:

$$\mathcal{D}(\tau, M, \nu) = \delta(\tau) - \nu(\mathcal{D}_\tau) \tag{1}$$

As for fixed priority further steps should be taken if, at some point, several tasks have the same deadline on a given processor. We deal with this in the next paragraph.

Dealing with equal priorities. In the case where several processes have the same priority at some point (be it fixed or dynamic) and if this priority makes them eligible for execution, the scheduler has at least three options:

– a deterministic choice with a second criterion, e.g. a FIFO choice on the list of processes (this may require to extend NewFlow from states to runs which causes no theoretical problem) ;
– a random choice between the processes with the same priority;
– time sharing between the processes with the same priority (*round-robin*).

The deterministic case is easily handled by an integration of the second choice criterion in the computation of the Flow function.

Since our model does not support probabilistic firing of transitions, the case of a random choice will be treated as a non-deterministic choice. In this case, the scheduler is not deterministic anymore: all possible outcomes are given by the function NewFlow as the set of all possible values of the function Flow, each corresponding to one of the possible choice of the scheduler.

Finally, in the Round-Robin scheduling policy, each task is given a small processor time slice (*quantum*), one after the other and repeatedly, leading to a pseudo-parallelism. This leads to a significant number of preemptions, which we do not want to explicitly model since it would cause the number of states to analyze to drastically increase. Hence, we model this policy by using a *fluid* approach illustrated by the chronogram in Figure 3. It represents the concurrent execution of two tasks τ_1 and τ_2 on the same processor, with the same priority. The task τ_1 is periodic with a period of 4 time units. Its execution time is 2 time units. The task τ_2 is periodic with period 8 and its execution time is 3. Finally the time quantum is $d = 0.5$.

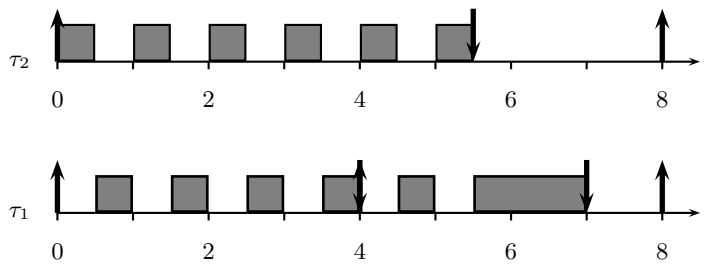

Fig. 3. Chronogram of the execution of two tasks scheduled on the same processor using Round-Robin

By letting d tend towards 0, we can model the scheduling policy by assuming that the processor is perfectly shared between all scheduled tasks. If n tasks are scheduled using Round-Robin, we consider that they are executed in full

parallelism at speed $\frac{1}{n}$. This is illustrated in Figure 4. The smaller the time quantum in comparison to the execution times of tasks, the closer to reality our model is. For $d > 0$, this is however neither an over-approximation nor an under-approximation of the real behavior: each time a Roud-Robin execution ends, the model approximates the real execution time of the stopped tasks by at most d time units.

Henceforward, we consider the "fluid" approach to Round-Robin.

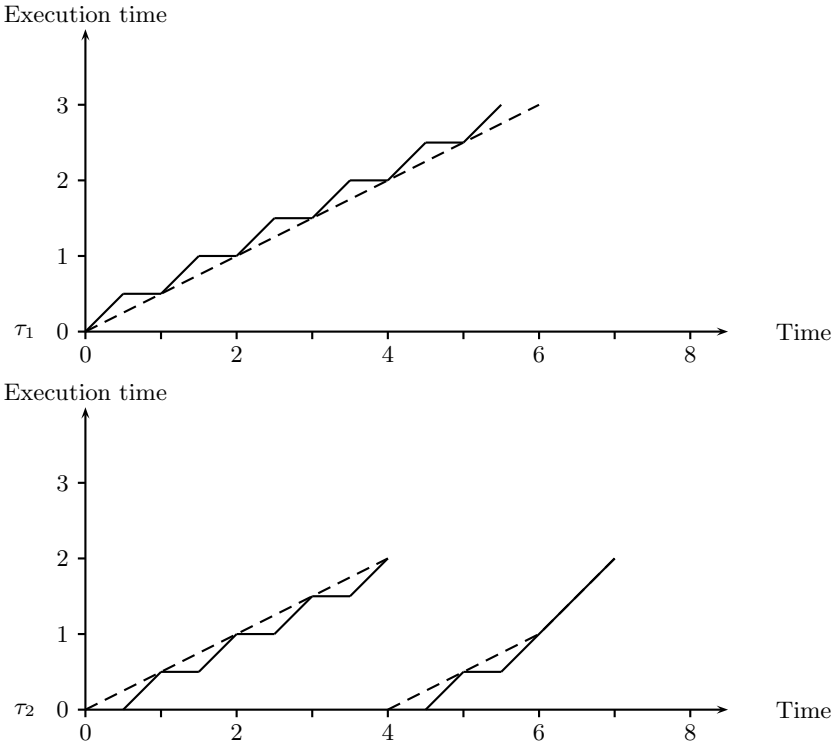

Fig. 4. Execution of the two tasks in Figure 3 with respect to time: real (plain) and model (dashed)

Hence, the value of the function Flow on t is given by the following: if $\gamma(t)$ has the greatest priority on its processor, then $\mathsf{Flow}(t) = \frac{1}{n}$, where n is the number of tasks sharing the same priority as $\gamma(t)$ on the processor $\Pi(\gamma(t))$ and $\mathsf{Flow}(t) = 0$ otherwise.

End of tasks. Note that the preemption of a task represented by a transition t can occur even when the transition t has met its upper bound. Indeed, a transition must fire when its valuation reaches its upper bound but firing transitions takes no time so any enabled transition, including t, may fire.

This is consistent with the real behavior of the scheduler for which a task may be preempted when it only has a call to some `end_of_task()` primitive left to do.

3 State-Class Graph

In this section, we show how to compute a first abstraction to analyze our models. This is based on the state-class graph method by Berthomieu, Diaz and Menasche (Berthomieu and Menasche 1983, Berthomieu and Diaz 1991) as well as on our first extension of that method for stopwatch Petri net models presented in (Lime and Roux 2003, Roux and Lime 2004).

Even if the net is bounded and as for all timed models, the explicit enumeration of all the reachable states of *Scheduling*-TPNs is forbidden by the use of a dense representation of time. Valuations may indeed take an infinite number of different values, leading to an infinity of states in the semantics (so-called *concrete* states).

A solution to this problem is to consider symbolic states which gather concrete states of the nets bound by some equivalence relation. By a careful choice of the equivalence relation, we can obtain a finite number of equivalence classes and guarantee that the graph of transitions between reachable classes preserves the properties of interest of the graph of transitions between reachable concrete states. Among these abstraction techniques are found: the region graph (Alur and Dill 1994), the zone graph (Larsen *et al.* 1995) and the state-class graph.

The state-class graph is historically the first one and, in opposition to the region and zone graphs proposed first for timed automata (Alur and Dill 1994) and then adapted to time Petri nets (Gardey *et al.* 2006), it was designed from the start to analyze time Petri nets (and its applicability to timed automata is an open problem). The region graph is an inefficient abstraction and mostly used as a theoretical tool. The zone graph however could also be used and it is actually implemented in our tool for the verification of TPN Romeo. The choice of the state-class graph is here mainly made for the sake of simplicity since it avoids to deal with the so-called k-extrapolations which ensure the finiteness of the zone graph.

In (Lime and Roux 2003, Roux and Lime 2004), we have extended the state-class graph method to take stopwatches into account. In the following, we show how to deal with the specific theoretical problems coming from our modeling of Round-Robin (multi-rate continuous variables) and Earliest Deadline First (unstability of symbolic states with respect to the activity function) policies.

3.1 State-class

Informally speaking, a state-class gathers all concrete states reached by a particular sequence of transitions (regardless of the timings of their firings). As a consequence, all concrete states in a state-class share the same marking.

Definition 3 (State-Class). *A* state class *of a time Petri net $\mathcal{N}$ is a pair (M, D) where M is a marking of $\mathcal{N}$ and D is a convex polyhedron called the* firing domain *and described by:*

$$A\Theta \leq B$$

If n is the number of transitions enabled by M, there exists $m \in \mathbb{N}$ s.t. A is a $m \times n$ matrix, B is a vector of dimension m, and Θ is the variable vector of dimension n.

In the class $C = (M, A\Theta \leq B)$, the Θ vector gives, for each enabled transition, the time remaining until it fires. This time is relative to the date at which the class C was reached. Given a polyhedron D on n variables, we will denote by $[\![D]\!]$ the set of real vectors which are solutions of D and $[\![D]\!]_\theta$ the projection of this set on the variable θ.

When the function Flow only depends on the current marking, as in a "Fixed Priority for all processors" setting, its value, for each enabled transition, is the same for all the concrete states of a given class C. Thus, we denote by $\mathsf{Flow}(C, t)$ this common value on an enabled transition t.

This is not the case anymore with an EDF scheduling policy, where Flow also depends on the clock valuation. Consider the *Scheduling*-TPN in Fig. 1. If we fire t_2 at date 1 then we reach a state s, with $\mathcal{D}(\tau_1, s) = 9$ and $\mathcal{D}(\tau_2, s) = 8$. So, τ_2 is selected by the scheduler. Now, if we fire t_2 at time 3, then the reached state s' is such that $\mathcal{D}(\tau_1, s') = 7$ and $\mathcal{D}(\tau_2, s') = 8$. This time, τ_1 is the task selected by the scheduler.

However, as we said above, all the states obtained by firing t_2 belong to the same state class C. So, given a transition t and a state s in C, $\mathsf{Flow}(t)$ is not unique. As a consequence, we need to perform some extra partitioning of the state-classes, so that the uniqueness of Flow on the partitions is enforced.

A correct partition with respect to Flow is obtained by adding extra variables modeling the deadlines and adding extra inequalities in the domain which enforce a total order on the firing times of these deadlines.

In subsection 2.3, for each task τ, we have added a transition $\mathcal{D}_\tau$ to model the time elapsed since the start of the task. With an abuse of notations, the corresponding variable in the domains of state classes will also be called $\mathcal{D}_\tau$. Since the bounds of the time interval of $\mathcal{D}_\tau$ are equal to $\delta(\tau)$, this variable exactly represents the time remaining until the expiration of the deadline of task τ.

So, we can add extra inequalities in the firing domain between these variables. Given a class C and two tasks τ and τ' such that the possible dates for their deadlines are overlapping, we can partition C by duplicating it as a new class

C' and adding $\mathcal{D}_\tau \leq \mathcal{D}'_\tau$ to the domain of C and $\mathcal{D}_\tau \geq \mathcal{D}'_\tau$ to the domain of C'. If we assume that no other deadlines were overlapping, then, on both partition and given a transition, Flow has the same value for all states.

3.2 Initial state class and successors

The initial state class of a *Scheduling*-TPN $\mathcal{N}$, is $C_0 = (M_0, D_0)$, M_0 being the initial marking of $\mathcal{N}$ and $D_0 = \{\alpha(t_i) \leq \theta_i \leq \beta(t_i) | t_i \in \mathsf{enabled}(M_0)\}$.

For each concrete state (M, ν, Flow) in the initial class, and for each transition t enabled by M, $\mathsf{Flow}(t)$ is unique since all states in C_0 are obtained by letting time elapsed from $(M_0, \overline{0}, \mathsf{Flow}_0)$, which cannot change the relative order of deadlines. We will show that the proposed successor computation preserves this property and as a consequence, we denote by $\mathsf{Flow}(C, t)$ the common value of $\mathsf{Flow}(t)$ for each concrete state s in class C.

Now we can define when a transition can be fired from a given state class:

Definition 4 (Firable transition from a state-class). *Let $C = (M, D)$ be a state class of a Scheduling-TPN. A transition t_i is said to be* firable from C *iff:*

- *t_i is active,*
- *there exists a solution $(\theta_1^*, \ldots, \theta_n^*)$ of D s.t. $\forall j \in [1..n] - \{i\}$, s.t. t_j is active, $\frac{\theta_i^*}{\mathsf{Flow}(C,t_i)} \leq \frac{\theta_j^*}{\mathsf{Flow}(C,t_j)}$,*
- *if there exists τ s.t. $t_i = \mathcal{D}_\tau$, then no transition in $E(\tau)$ is firable from C.*

Given the initial class and the firability rule above, we can compute a state-class graph by applying the following successor computation: let $C = (M, D)$ be a state class and t_f a firable transition from C. Then, the successor $C' = (M', D')$ of C by t_f is given by the following rules:

- the new marking is computed as usual by $M' = M - {}^\bullet t_f + t_f{}^\bullet$;
- the new firing domain, D', also denoted by $\mathsf{Next}(D, t_f)$ is computed by the following algorithm:
  1. $\forall j \neq f$, s.t. t_j is active, addition of the constraints $\frac{\theta_f}{\mathsf{Flow}(C,t_f)} \leq \frac{\theta_j}{\mathsf{Flow}(C,t_j)}$,
  2. variable substitutions: for all active transition $t_j (j \neq f)$ in C, $\theta_j = \frac{\mathsf{Flow}(C,t_j)}{\mathsf{Flow}(C,t_f)} * \theta_f + \theta'_j$,
  3. elimination of variables corresponding to transitions disabled by the firing of t_f (thus including t_f),
  4. addition of inequalities relative to transitions newly enabled by the firing of t_f:
  $$\forall t_k \in \uparrow enabled(M, t_f), \alpha(t_k) \leq \theta'_k \leq \beta(t_k)$$
  5. for each task τ, s.t. $Sched(\Pi(\tau)) = EDF$ and that is beginning, *i.e.* s.t. $t_f \in B(\tau)$, if $\exists \tau'$ s.t. $\delta(\tau) \in [\![D']\!]_{\mathcal{D}_{\tau'}}$, then we duplicate the class. One of the instances receives the additional inequality $\mathcal{D}_\tau \leq \mathcal{D}_{\tau'}$ and the other $\mathcal{D}_\tau \geq \mathcal{D}_{\tau'}$.

The constraints added in step 1 mean that t_f has been selected for firing, so it will fire before any other active transition. Adding these constraints never leads to an empty firing domain since t_f is firable and therefore meets the conditions in definition 4.

The variable substitutions in step 2 model the time elapsing. The new origin of time for the successor is chosen as the firing date of t_f. So all active transitions progress according to the delay after which t_f is fired (since the firing of the transition that lead to the class C). Classically, in *TPNs*, the coefficient before θ_f is 1 since all clocks progress at rate 1(Berthomieu and Diaz 1991). In *Scheduling*-TPNs with a fixed priority policy but without Round-Robin, we extended it to 0 (preempted tasks) or 1 (active tasks) (Roux and Déplanche 2002, Lime and Roux 2003). Now, the "fluid" approach to the modelling of Round-Robin that we chose leads to more complex rates, given by the Flow function.

Elimination of variables in step 3 may be done using the Fourier-Motzkin method (Dantzig 1963). According to the semantics, we know that only variables associated to enabled transitions are relevant to describe the state of the net. So variables relative to disabled transitions do not need to be explicitly kept and are eliminated. Note that since we added deadline transitions to the net, the variables relative to the deadlines of tasks τ ending with the firing of t_f ($t_f \in E(\tau)$) are eliminated at this step.

Similarly, when a task τ begins ($t_f \in B(\tau)$), its deadline transition becomes (newly) enabled. So the $\delta(\tau) \leq \mathcal{D}_\tau \leq \delta(\tau)$ inequalities are added to the firing domain at step 4, setting up a timer until the expiration of the deadline.

The last step realizes the partitioning when the possible dates of the deadline expiration of different tasks overlap. Note that when partitioning a class between two overlapping deadlines, the case where $\mathcal{D}_\tau = \mathcal{D}_{\tau'}$ belongs to both partitions. This reflects a non-deterministic choice made by the scheduler. Then both cases are considered in the analysis. If the choice is deterministic, then one of the added inequation should be strict (which causes no problem for the theory of time Petri nets nor of convex polyhedra) and if Round-Robin is chosen, then the class should be split in three: $\mathcal{D}_\tau < \mathcal{D}'_\tau$, $\mathcal{D}_\tau > \mathcal{D}'_\tau$ and $\mathcal{D}_\tau = \mathcal{D}'_\tau$.

Theorem 1 (Class uniqueness of Flow). *For each state class $C = (M, D)$ and for each transition t enabled by M, there exists $F \in \mathbb{Q}$, s.t. for all states $s = (M, \nu, \mathsf{Flow}) \in C$, $\mathsf{Flow}(t) = F$. We note $\mathsf{Flow}(C, t) = F$.*

Proof. For the Fixed Priority policy, Flow only depends on the marking. Since all states in a class share the same marking, they also have the same Flow function.

For the Earliest Deadline First policy, Flow also depends on the relative order of (the expiration of) deadlines. The added constraints on variables $\mathcal{D}_\tau$ ensure that all states in a class have the same deadline order, which entails that they have the same Flow function.

3.3 Examples

Round-Robin. We now compute the state-class graph of Example 2, given in Figure 2. The result is given in Figure 5.

The initial class is:

$$C_0 = \begin{cases} \{p_1, p_2, p_3, p_4\} \\ 4 \leq \theta_1 \leq 4 \\ 8 \leq \theta_3 \leq 8 \\ 2 \leq \theta_2 \leq 2 \\ 3 \leq \theta_4 \leq 3 \end{cases}$$

t_2 and t_4 share the same processor with the same priority and thus are both active. The processor is ideally shared between them: $\mathsf{Flow}(t_2) = \mathsf{Flow}(t_4) = \frac{1}{2}$. t_1 and t_2 are both firable. We fire t_2. The variable substitutions are done as follows:

$$C_1' = \begin{cases} \{p_1, p_3, p_4\} \\ 4 \leq \theta_1' + 2\theta_2 \leq 4 \\ 8 \leq \theta_3' + 2\theta_2 \leq 8 \\ 2 \leq \theta_2 \leq 2 \\ 3 \leq \theta_4' + \frac{1}{2}2\theta_2 \leq 3 \end{cases}$$

And the class finally obtained is:

$$C_1 = \begin{cases} \{p_1, p_3, p_4\} \\ 0 \leq \theta_1 \leq 0 \\ 4 \leq \theta_3 \leq 4 \\ 1 \leq \theta_4 \leq 1 \end{cases}$$

t_1 is then fired in 0 time unit, which gives the following class:

$$C_2 = \begin{cases} \{p_1, p_2, p_3, p_4\} \\ 4 \leq \theta_1 \leq 4 \\ 4 \leq \theta_3 \leq 4 \\ 2 \leq \theta_2 \leq 2 \\ 1 \leq \theta_4 \leq 1 \end{cases}$$

Firing first t_1 then t_2 leads to this class. Now, only t_4 is firable. Again, t_2 and t_4 are executing concurrently on the same processor with the same priority. So, we have:

$$C_3' = \begin{cases} \{p_1, p_2, p_3\} \\ 4 \leq \theta_1' + 2\theta_4 \leq 4 \\ 4 \leq \theta_3' + 2\theta_4 \leq 4 \\ 2 \leq \theta_2' + \frac{2}{2}\theta_4 \leq 2 \\ 1 \leq \theta_4 \leq 1 \end{cases} \quad C_3 = \begin{cases} \{p_1, p_2, p_3\} \\ 2 \leq \theta_1 \leq 2 \\ 2 \leq \theta_3 \leq 2 \\ 1 \leq \theta_2 \leq 1 \end{cases}$$

Now, t_2 is the only firable transition and it has the processor for itself. The class obtained after firing t_2 is:

$$C_4 = \begin{cases} \{p_1, p_3\} \\ 1 \leq \theta_1 \leq 1 \\ 1 \leq \theta_3 \leq 1 \end{cases}$$

Finally firing t_1 and t_3 in any order straightforwardly leads back to the initial class. Figure 5 gives the resulting graph.

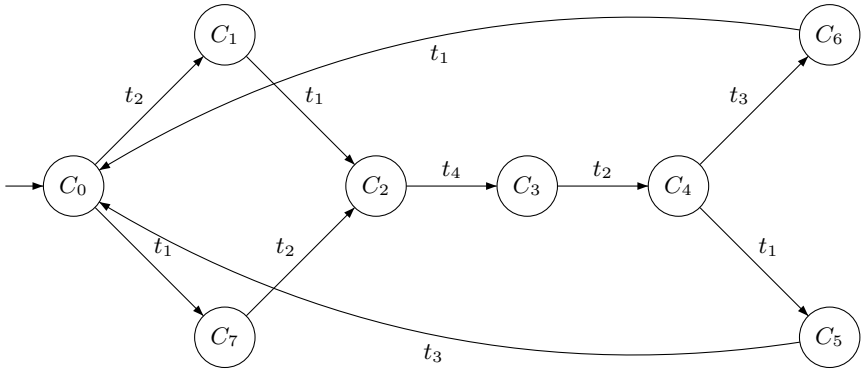

Fig. 5. State class graph of the net in Figure 2

Earliest Deadline First. We now compute the state-class graph of the net presented in Figure 1. In this example, we do not explicitly add the deadline transitions but the variables are added in the domain exactly as if they were present.

The initial class is:

$$C_0 = \begin{cases} \{p_1, p_2\} \\ 10 \leq \theta_1 \leq 10 \\ 1 \leq \theta_2 \leq 3 \\ 10 \leq \mathcal{D}_{\tau_1} \leq 10 \end{cases}$$

We have $\mathsf{Flow}(t_1) = \mathsf{Flow}(t_2) = 1$. The only firable transition is t_2. Its firing gives:

$$C_1 = \begin{cases} \{p_1, p_3, p_4\} \\ 7 \leq \theta_1 \leq 9 \\ 3 \leq \theta_3 \leq 3 \\ 2 \leq \theta_4 \leq 2 \\ 7 \leq \mathcal{D}_{\tau_1} \leq 9 \\ 8 \leq \mathcal{D}_{\tau_2} \leq 8 \\ 0 \leq \theta_1 - \mathcal{D}_{\tau_1} \leq 0 \end{cases}$$

8 t_2 Tf 7.80Td 255.973(a)-5.9.961 2.880-3 429.961 2.880T /R10933257602Td [8d [(1-11.8801Td 0)-5.8887

12.9tS T

$\theta_4 \leq 2$

7

$, p$ 4

In the class C_1', we have $\mathsf{Flow}(t_3) = 1$ and $\mathsf{Flow}(t_4) = 0$. Only t_3 is firable. So we fire t_3 and obtain:

$$C_2 = \begin{cases} \{p_1, p_4\} \\ 4 \leq \theta_1 \leq 6 \\ 2 \leq \theta_4 \leq 2 \\ 5 \leq \mathcal{D}_{\tau_2} \leq 5 \end{cases}$$

t_3 ends τ_1, so its firing allows us to remove the inequalities relative to $\mathcal{D}_{\tau_1}$. In C_2, $\mathsf{Flow}(t_4) = 1$ and t_4 is the only firable transition. The successive firings of t_4 and t_1 straightworfardly lead back to C_0.

In the class C_1'', we have $\mathsf{Flow}(t_3) = 0$ and $\mathsf{Flow}(t_4) = 1$. Only t_4 is firable. So, we fire t_4, which gives:

$$C_3 = \begin{cases} \{p_1, p_3\} \\ 5 \leq \theta_1 \leq 7 \\ 3 \leq \theta_3 \leq 3 \\ 5 \leq \mathcal{D}_{\tau_1} \leq 7 \\ 0 \leq \theta_1 - \mathcal{D}_{\tau_1} \leq 0 \end{cases}$$

As before, t_4 ends τ_2 so with its firing, we remove $\mathcal{D}_{\tau_2}$ from the firing domain of C_3. Then the sequence t_3, t_1 leads back to the initial class.

Finally, we obtain the (non-deterministic) graph in Figure 6.

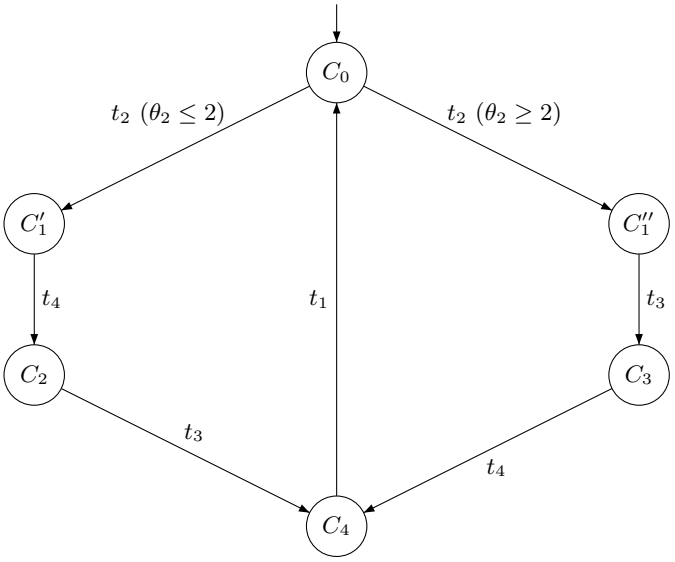

Fig. 6. State class graph of the net in Figure 1.

4 State Class Hybrid Automaton

In this section, we present a method for computing the state space of a *Scheduling*-TPN as a linear hybrid automaton (LHA)(Alur *et al.* 1995). We show the soundness of the computation by proving that this LHA is time-bisimilar to the initial *Scheduling*-TPN. We then show how to obtain, with a much faster DBM-based over-approximating method, a LHA which is also time-bisimilar to the *Scheduling*-TPN. But first, let us recall the definition of linear hybrid automata.

4.1 Linear hybrid automata

A hybrid automaton is basically a finite automaton equipped with continuous variables whose evolution is dictated by the current location.

We actually consider a restricted form of linear hybrid automata consisting of timed automata (Alur and Dill 1994) augmented with derivatives of continuous variables given as integer constants in each location.

Definition 5 (Linear Hybrid Automaton). *A* linear hybrid automaton *is a 7-tuple* $(L, l_0, X, A, E, Inv, Dif)$ *where*

- L *is a finite set of* locations*,*
- l_0 *is the* initial location*,*
- X *is a finite set of real-valued variables,*
- A *is a finite set of* actions*,*
- $E \subset L \times \mathcal{C}(X) \times A \times 2^X \times X^X \times L$ *is a finite set of edges. If* $e = (l, \delta, \alpha, R, \rho, l') \in E$*,* e *is the edge between locations* l *and* l'*, with the* guard δ*, the* action α*, the set of variables to* reset R *and the* clock assignment *function* ρ*.*
- $Inv \in \mathcal{C}(X)^L$ *maps an* invariant *to each location,*
- $Dif \in (\mathbb{N}^X)^L$ *maps an* activity *to each location,* $\dot{X}$ *being the set of derivatives of the variables w.r.t. time.* $\dot{X} = (Dif(l, x))_{x \in X}$*.*

For short, given a location l, a continuous variable x and $n \in \mathbb{N}$, we will denote $Dif(l, x) = n$ by $\dot{x} = n$ when the considered location is not ambiguous.

Definition 6 (Semantics of a LHA). *The semantics of a LHA* H *is defined as a TTS* $\mathcal{S}_H = (Q, Q_0, \rightarrow)$ *where* $Q = L \times (\mathbb{R}^+)^X$*,* $Q_0 = (l_0, \overline{0})$ *is the initial state and* $\rightarrow$ *is defined, for* $a \in A$ *and* $t \in \mathbb{R}^+$*, by:*

- discrete *transitions:* $(l, \nu) \xrightarrow{a} (l', \nu')$ *iff* $\exists (l, \delta, a, R, \rho, l') \in E$ *such that*
$$\begin{cases} \delta(\nu) = \texttt{true}, \\ \nu' = \nu[R \leftarrow 0][\rho], \\ Inv(l')(\nu') = \texttt{true} \end{cases}$$
- continuous *transitions:* $(l, \nu) \xrightarrow{t} (l, \nu')$ *iff* $\begin{cases} \nu' = \nu + \dot{X} * t, \\ \forall t' \in [0, t], Inv(l)(\nu + \dot{X} * t') = \texttt{true} \end{cases}$

4.2 State class hybrid automaton

Following the idea of (Lime and Roux 2006*a*) for classical time Petri nets, we extend the notion of state classes with information about the continuous variables that are required to describe the class. Then, we compute the reachability graph of these extended state classes with an adequate convergence criterion. Finally, we syntactically compute the hybrid automaton from the extended state-class graph.

First, let us define extended state-classes:

Definition 7 (Extended state-class). *An extended state class is a 4-tuple* $(M, D, \chi, trans)$*, where* M *is a marking,* D *a firing domain,* χ *a set of continuous variables and* $trans \in (2^T)^\chi$ *a mapping of variables to sets of transitions.*

The variables in χ measure the cumulative time during which the transitions associated to by $trans$ have been active since they have been enabled.

Given an extended state class $C = (M, D, \chi, trans)$ and a firable transition t_f, the successor $C' = (M', D', \chi', trans')$ of C obtained by firing t_f is given by the following algorithm:

1. M' and D' are computed as in section 2,
2. for each variable x in χ, the disabled transitions are removed from $trans(x)$,
3. the variables whose image by $trans$ is empty are removed from χ,
4. if there are newly enabled transitions by the firing of t, two cases can occur:
   - there exists a variable x whose value is zero[3]. Then, we simply add the newly enabled transitions to $trans(x)$,
   - There is no such variable. Then we need to create a new one, x_i associated to the newly enabled transitions. The index, i, is chosen as the smallest available index w.r.t. the variables in χ. We add x_i to χ and $trans(x_i)$ is the set of newly enabled transitions.
5. if, for a given variable x, the value of $\mathsf{Flow}(C, t)$ is not the same for all the transitions t in $trans(x)$, then we create as many variables as there are different values minus one (the considered variable). Then, we remap all the transitions to these variables via $trans$ so that for each variable x_j, $\mathsf{Flow}(C)$ is constant on $trans(x_j)$.

By applying theses rules, the extended state-class graph is computed by generating all the successors of the initial state-class iteratively (breadth-first for instance). The chosen convergence criterion is χ*-similarity*, with an inclusion check:

Definition 8 (χ-similarity). *Two extended state classes* $C = (M, D, \chi, trans)$ *and* $C' = (M', D', \chi', trans')$ *are* χ-similar*, and we denote it by* $C \approx C'$*, iff they*

[3] This means that no time could elapse since the creation of x. It is fairly easy to check from D and D' if some time could elapse when firing t_f. So, it is easy to check inductively if some time could elapse since the creation of x.

have the same markings and the same number of continuous variables which are mapped to the same transitions:

$$C \approx C' \Leftrightarrow \begin{cases} M = M', \\ |\chi| = |\chi'|, \\ \forall x \in \chi, \exists x' \in \chi', \ s.t. \\ \begin{cases} trans(x) = trans'(x'), \\ \forall t \in trans(x), \forall t' \in trans(x)', \mathsf{Flow}(C,t) = \mathsf{Flow}(C',t') \\ \ and\ x = 0 \Leftrightarrow x' = 0 \end{cases} \end{cases}$$

Definition 9 (Inclusion of state classes). *An extended state class* $C' = (M', D', \chi', trans')$ *is* included *in an extended state class* $C = (M, D, \chi, trans)$ *iff* C *and* C' *are* χ*-similar and* $[\![D']\!] \subset [\![D]\!]$*. This is denoted by* $C' \subset C$*.*

So, when two classes are χ-similar and one is included in the other, we stop the exploration of the current branch. If there is no inclusion, then we loop anyway but continue the computation of the successors of the states that are not in the intersection of the two domains.

We write the extended state class graph as the following timed transition system: $\Delta'(\mathcal{T}) = (C^{ext}, C_0, \rightarrow^{ext})$ defined by:

- $C^{ext} \subseteq \mathbb{N}^P \times \mathbb{R}^T \times 2^X \times (2^T)^X$, X being the set of all continuous variables,
- $C_0 = (M_0, D_0, \chi_0, trans_0)$, where M_0 is the initial marking, $D_0 = \{\alpha(t_i) \leq \theta_i \leq \beta(t_i) | \, t_i \in \mathsf{enabled}(M_0)\}$, $\chi_0 = \{x_0, x_1\}$ and $trans_0 = \{(x_k, \{t \in \mathsf{enabled}(()M_0) | \mathsf{Flow}_0(t) = r_k\})\}$ with r_k ranging over the different rates of transitions enabled by M_0,
- $\rightarrow^{ext} \in C^{ext} \times T \times C^{ext}$ is the transition relation defined by the algorithm of definition 8.

And now, using this timed transition system we give the definition of the state-class hybrid automaton.

Definition 10 (State-Class Hybrid Automaton). *The state-class hybrid automaton* $\Delta(\mathcal{T}) = (L, l_0, X, A, E, Inv, Dif)$ *is defined from the extended state-class graph by:*

- L*, the set of locations, is the set of the extended state classes* C^{ext}*,*
- l_0 *is the initial state class* $(M_0, D_0, \chi_0, trans_0)$*,*
- $X = \bigcup_{(M,D,\chi,trans) \in C^{ext}} \chi$ *the set of all continuous variables,*
- $A = T$ *is the set of transitions,*
- E *is the set of edges, defined as follows:*

$$\forall C_i = (M_i, D_i, \chi_i, trans_i), C_j = (M_j, D_j, \chi_j, trans_j) \in C^{ext},$$
$$\exists C_i \overset{t}{\rightarrow}^{ext} C_j \Leftrightarrow \exists (l_i, \delta, a, R, \rho, l_j) \ s.t. \begin{cases} \delta = (trans_i^{-1}(t) \geq \alpha(t)), \\ a = t, \\ R = trans_j^{-1}(\uparrow \mathsf{enabled}(M_i, t)), \\ \forall x \in \chi_i, x' \in \chi_j, \\ \ s.t.\ trans_j(x') \subset trans_i(x) \\ \ and\ x' \notin R, \rho(x) = x' \end{cases}$$

– $\forall C \in C^{ext}, Inv(C) = \bigwedge_{x \in \chi, t \in trans(x)} (x \leq \beta(t))$.
– $\forall C \in C^{ext}, \forall x \in \chi, Dif(C,x)$ *is the common value of* $\mathsf{Flow}(C)$ *on all the transitions of* $trans(x)$.

As an example, Figure 7 shows a *Scheduling*-TPN modeling two periodic tasks running on the same processor and synchronized by a semaphore. Figure 8 shows the corresponding state class LHA. The semaphore is simply modeled by a place *Semaphore*. The actions *semP* and *semV* are then obtained respectively by *waiting for a token from Semaphore* and *adding a token in Semaphore*.

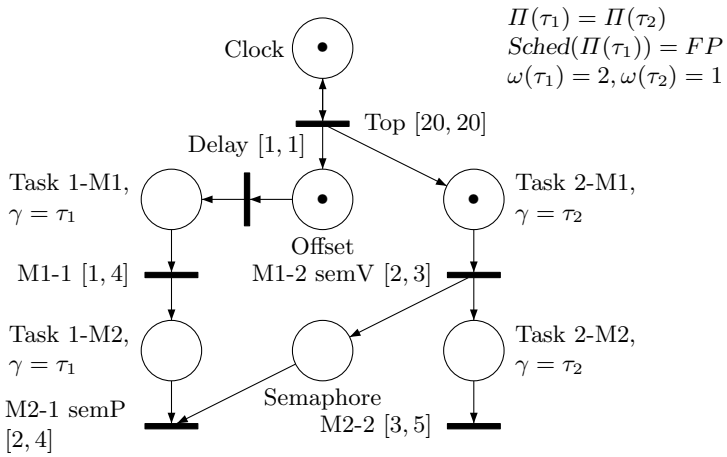

Fig. 7. *Scheduling*-TPN modeling two tasks sharing a processor and synchronized by a semaphore

4.3 Termination of the algorithm

As for time Petri nets, reachability is undecidable for *Scheduling*-TPNs as well as for LHA. For bounded TPN and timed automata, it is actually decidable. Reachability is however undecidable even for bounded *Scheduling*-TPNs. Since our method is based on the computation of an extended reachability graph, the computation of the state class hybrid automaton is not guaranteed to finish, which is inherent to that class of models.

Note that a structural translation from *Scheduling*-TPNs to LHA along the lines of (Cassez and Roux 2006) should be feasible and decidable but the analysis of the obtained LHA would be very inefficient. Our objective is to obtain an LHA which can be efficiently analyzed as we will show in the next sections.

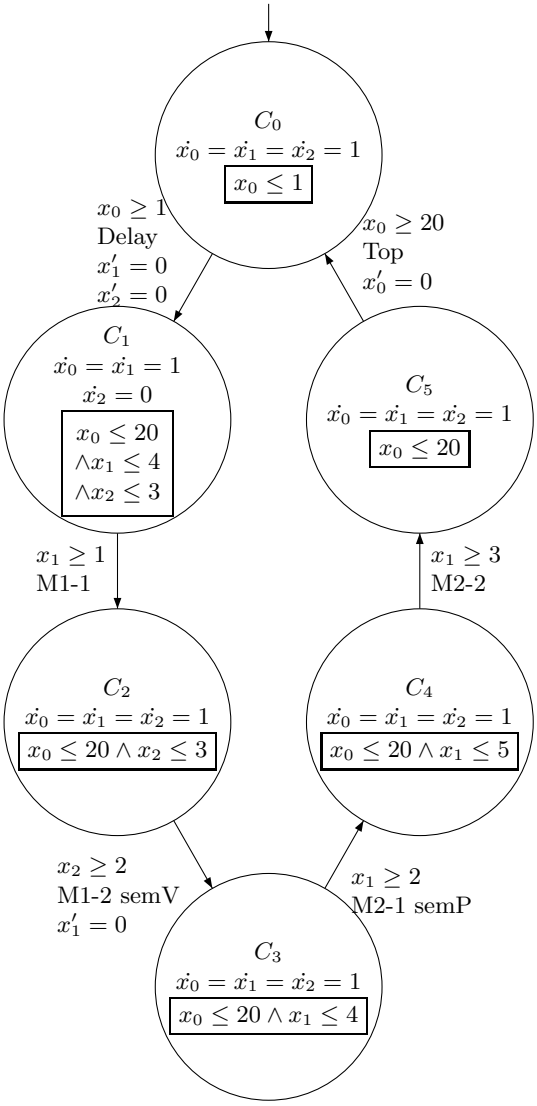

Fig. 8. State-class LHA of the net in Figure 7. The initial location is C_0.

4.4 Soundness of the translation

In order to prove the soundness of this expression of the state space of a *Scheduling*-TPN, we will show in theorem 2 that the *Scheduling*-TPN and its state class LHA are time-bisimilar.

Theorem 2 (Bisimulation). *Let $Q_{\mathcal{T}}$ be the set of states of the Scheduling-TPN $\mathcal{T}$ and $Q_{\mathcal{A}}$ the set of states of the state-class hybrid automaton $\mathcal{A} = (L, l_0, X, A, E, Inv, Dif)$. Let $\mathcal{R} \subset Q_{\mathcal{T}} \times Q_{\mathcal{A}}$ be a binary relation such that $\forall s = (M_{\mathcal{T}}, \nu_{\mathcal{T}}, \mathsf{Flow}) \in Q_{\mathcal{T}}, \forall a = (l, \nu_{\mathcal{A}}) \in Q_{\mathcal{A}}, s\mathcal{R}a \Leftrightarrow M_{\mathcal{T}} = M_{\mathcal{A}}$ where $M_{\mathcal{A}}$ is the marking of the extended state-class l and $\forall t \in \mathsf{enabled}(M_{\mathcal{T}})$, $\exists x_t \in X, \nu_{\mathcal{T}}(t) = \nu_{\mathcal{A}}(x_t)$ and $\dot{x_t} = \mathsf{Flow}(t)$.*

$\mathcal{R}$ is a bisimulation.

Proof. Let $s = (M_{\mathcal{T}}, \nu_{\mathcal{T}}, \mathsf{Flow}) \in Q_{\mathcal{T}}, a = (l, \nu_{\mathcal{A}}) \in Q_{\mathcal{A}}$ and $s\mathcal{R}a$. Then $\forall t \in \mathsf{enabled}(M_{\mathcal{T}}), \exists x_t \in X, \nu_{\mathcal{T}}(t) = \nu_{\mathcal{A}}(x_t)$ and $\dot{x_t} = \mathsf{Flow}(t)$.

1. Let us suppose that the *Scheduling*-TPN can let $d \in \mathbb{R}^+$ time units elapse: $s \xrightarrow{d} s'$. That means that $\forall t \in \mathsf{enabled}(M_{\mathcal{T}})$, $\nu_{\mathcal{T}}(t) + \mathsf{Flow}(t).d \leq \beta(t)$. Then, $\nu_{\mathcal{A}}(x_t) + \dot{x_t}.d \leq \beta(t)$ and so by definition of $Inv(l)$, $Inv(l)(\nu_{\mathcal{A}} + \dot{X}.d')$ is true $\forall d' \leq d$. Therefore, the state-class LHA $\mathcal{A}$ can let time elapse during d time units: $a \xrightarrow{d} a'$. Since the *Scheduling*-TPN stays in the same extended state class, and the state-class LHA in the same location, the activity Flow and $\dot{X}$ do not change and finally $s'\mathcal{R}a'$.
2. Let us suppose that the *Scheduling*-TPN can fire the transition $t \in T : s \xrightarrow{t} s'$ with $s' = (M'_{\mathcal{T}}, \nu'_{\mathcal{T}}, \mathsf{Flow}')$. By definition of the state-class hybrid automaton, there exists an edge $e = (l, \delta, t, R, \rho, l')$. That means that $\alpha(t) \leq \nu_{\mathcal{T}}(t)$. So, $\alpha(t) \leq \nu_{\mathcal{A}}(x_t)$ and by definition of the guard δ, $\delta(\nu_{\mathcal{A}})$ is verified. Therefore the state class LHA $\mathcal{A}$ can take the edge $e : a \xrightarrow{t} a'$. By definition of l', the marking $M'_{\mathcal{A}}$ of the extended state-class l' is the same as the new marking $M'_{\mathcal{T}}$ of the *Scheduling*-TPN. Let t' be a transition in $\mathsf{enabled}(M'_{\mathcal{T}})$.
   If t' is newly enabled, a new clock is created or t' is associated to clock $x_{t'}$ whose value is 0 and $\dot{x_{t'}} = \mathsf{Flow}'(t')$. If t' is not newly enabled and if all transitions associated to its clock have the same activity, which is equal to $\mathsf{Flow}'(t')$, then $\dot{x_{t'}}$ is set accordingly to the activity of t': $\dot{x_{t'}} = \mathsf{Flow}'(t')$. Else, if all transitions associated to the clock of t' do not have the same activity value, a new clock $x_{t'bis}$ to which is associated t' is created with $\dot{x}_{t'bis} = \mathsf{Flow}'(t')$. Finally, $s'\mathcal{R}a'$.
3. Let us suppose that the state-class LHA can let $d \in \mathbb{R}^+$ time units elapse: $a \xrightarrow{d} a'$. That means that $Inv(l)(\nu_{\mathcal{A}} + \dot{X}.d)$ is true. So, by definition of $Inv(l)$, $\forall x \in X, \forall t \in trans(x)$, $\nu_{\mathcal{A}}(x) + \dot{x}.d \leq \beta(t)$, which is equivalent to $\forall x \in X, \forall t \in trans(x)$, $\nu_{\mathcal{T}}(t) + \mathsf{Flow}(t).d \leq \beta(t)$. Since $\bigcup_{x \in X} trans(x) = \mathsf{enabled}(M_{\mathcal{T}})$, we have finally $\forall t \in \mathsf{enabled}(M_{\mathcal{T}}), \nu_{\mathcal{T}}(t) + \mathsf{Flow}(t).d \leq \beta(t)$, which means that $\mathcal{T}$ can let d time units elapse: $s \xrightarrow{d} s'$. Since the state-class LHA stays in the same locality and then the *Scheduling*-TPN stays in the same extended state-class, the conditions on Flow and $\dot{X}$ do not change and finally $s'\mathcal{R}a'$.

4. Let us suppose that the state class LHA can take the edge $e = (l, \delta, t, R, \rho, l')$: $a \xrightarrow{t} a'$. That means that t is enabled by $M_{\mathcal{A}} = M_{\mathcal{T}}$ and that $\delta(\nu_{\mathcal{A}})$ is true. So, by definition of δ, $\nu_{\mathcal{A}}(x_t) \geq \alpha(t)$ and then $\nu_{\mathcal{T}}(t) \geq \alpha(t)$: t is therefore firable for $\mathcal{T}$. As in point 2, $s'\mathcal{R}a'$ by construction.

4.5 Number of continuous variables

As mentioned before, the number of continuous variables is a critical concern for the computation of the state-space of formal models. So, in this method we take great care to keep the number of those variables as low as possible. Modeling with *Scheduling*-TPNs roughly requires the same number of continuous variables[4] as a direct modeling as a product of LHA minus the possible variables of the scheduler. For instance, the basic modeling of a periodic task requires at least two variables for both models: one for the periodic activation and one for the progress of the task itself.

However, in the product of LHA, all variables are always used to define the state in the system whereas with *Scheduling*-TPNs, only the valuations of enabled transitions need to be considered. That means, for example, that when a periodic task is waiting for its periodical activation, only the variable associated with the activator is required.

As a consequence, we create these variables "on demand", *i.e.* when transitions are newly enabled. Moreover, when creating a new variable, we reuse those that are no longer used (because every associated transition has been disabled), by always choosing the first available index. Furthermore, we use only one variable for transitions for which valuations are necessarily equal, *i.e.* transitions that are simultaneously enabled, as long as they are running together, with the same evolution rate. Note that this configuration occurs fairly often.

Applying this policy for the creation of variables allows us to obtain a state-class hybrid automaton with a fairly low number of variables, in practical cases.

A maximum bound on the number of continuous variables is the maximum number of simultaneously enabled transitions in the *Scheduling*-TPN.

4.6 DBM over-approximation

As we have seen in definition 3, the domain of state classes is represented by a convex polyhedron. When dealing with timed systems where all variables evolve at rate one (e.g. time Petri nets or timed automata), the constraints on the continuous variables have the special form: $\theta_i - \theta_j \leq d_{ij}$ or $-d_{0i} \leq \theta_i \leq d_{i0}$. This allows the recourse to a more efficient data structure: the matrix whose coefficients are d_{ij}, which is called a Difference Bound Matrix (DBM) (Berthomieu and Menasche 1983, Dill 1989).

With more complex dynamics, like those in which we are interested in this paper, the special form of the constraints is lost and DBMs cannot be used for an

[4] In a *Scheduling*-TPN, these are actually the transition valuations.

exact computation of the state-space. We can however approximate each polyhedron by the smallest containing DBM. This leads to an over-approximation as the DBM contains states that are not part of the actual state-space of the *Scheduling*-TPN. So, an analysis directly based on the over-approximated state-space would be somehow "pessimistic" (e.g. it would declare that the model of the system is not schedulable whereas it actually is) but safe in the sense that if a bad state is declared not reachable, then it really is not reachable.

The over-approximation of domains by DBMs can be used to compute extended state-classes. This may lead to additional locations in the state class hybrid automaton. But, as the guards and invariants are statically computed from the parameters of the *Scheduling*-TPN itself, these additional locations are actually not reachable in the LHA. As a consequence, the relation $\mathcal{R}$ of theorem 2 is also a bisimulation between the over-approximated state class hybrid automaton and the *Scheduling*-TPN:

Theorem 3 (Bisimulation). *Let $Q_{\mathcal{T}}$ be the set of states of the Scheduling-TPN $\mathcal{T}$ and $Q_{\mathcal{A}}$ the set of states of the DBM over-approximated state class hybrid automaton $\mathcal{A} = (L, l_0, X, A, E, Inv)$. Let $\mathcal{R} \subset Q_{\mathcal{T}} \times Q_{\mathcal{A}}$ be a binary relation such that $\forall s = (M_{\mathcal{T}}, \nu_{\mathcal{T}}) \in Q_{\mathcal{T}}, \forall a = (l, \nu_{\mathcal{A}}) \in Q_{\mathcal{A}}, s\mathcal{R}a \Leftrightarrow M_{\mathcal{T}} = M_{\mathcal{A}}$ where $M_{\mathcal{A}}$ is the marking of the extended state-class l and $\forall t \in \mathsf{enabled}(M_{\mathcal{T}})$, $\exists x_t \in X, \nu_{\mathcal{T}}(t) = \nu_{\mathcal{A}}(x_t)$ and $\dot{x_t} = \mathsf{Flow}(t)$.*

$\mathcal{R}$ is a bisimulation.

The proof is the same as for the exact computation. Indeed, to make this clearer, let us suppose that in location l there is an outgoing edge $e = (l, \delta, t, R, \rho, l')$ because t is firable in the approximated state class l whereas it is not in the corresponding exact state class. If we suppose that before reaching the location l, the behavior of the automaton was correct, then right at the entry in the class l the automaton is in a state $a = (M, \nu_{\mathcal{A}})$ which is in relation with some state $s = (M, \nu_{\mathcal{T}})$ of the *Scheduling*-TPN by $\mathcal{R}$. On the one hand, since t is actually not firable, that means that some other transition t' must be fired before it: $\alpha(t) - \nu_{\mathcal{T}}(t) > \beta(t') - \nu_{\mathcal{T}}(t')$. So, by definition of $\mathcal{R}$, there exists x_t such that $\alpha(t) - \nu_{\mathcal{A}}(x_t) > \beta(t') - \nu_{\mathcal{T}}(t')$. Since, by definition of a *Scheduling*-TPN, $\beta(t') \geq \nu_{\mathcal{T}}(t')$, this gives $\alpha(t) - \nu_{\mathcal{A}}(x_t) > 0$. On the other hand, by definition of the guards of the state class LHA, "δ is true" is equivalent to $\alpha(t) - \nu_{\mathcal{A}}(x_t) \leq 0$. With the previous statement we can conclude that the guard δ is false and therefore that l' is not reachable.

As a conclusion, we can compute the state-class hybrid automaton with the fast DBM-based over-approximating algorithm. It may produce a few extra locations but the latter are not reachable and will be discarded during the HyTech analysis. It keeps a low cost for the translation in comparison to the verification cost. Moreover, we can benefit from the expressivity and ease of use of the *Scheduling*-TPN model. Finally, the state class LHA is, in general, easier to verify than a direct model using a product of LHA, since it has less continuous variables.

5 Verification

The analysis we perform on the state-class LHA has two goals: verify the schedulability of the system and verify its functional correctness. These formal verifications may be performed with HyTech. This tool carries out a symbolic analysis of LHA using polyhedra, called "regions". HyTech provides a number of functions to handle regions, including the computation of reachable states from a given region, the computation of successor states, existential quantification, convex hull and basic boolean operations (equality, emptiness, *etc.*).

We have compared the efficiency of our method, implemented in the Romeo (Gardey *et al.* 2005) tool, with a generic direct modeling using hybrid automata.

5.1 Direct modeling

This modeling requires one automaton per task and one per scheduler (and therefore per processor). The result is then a synchronized product *à la* Arnold-Nivat (Arnold 1994).

Figure 9 gives the generic automaton modeling a task τ_i periodic with the period T_i and whose execution time belongs to the interval $[\alpha_i, \beta_i]$. This model is inspired by the model of extended tasks in Osek/Vdx (OSEK 2001). Notably, message waiting is modeled by a $Waiting$ location. The synchronization mechanisms which produce the $release_i!$ and $waitEvent_i!$ events, and which react to the $sendEvent_i!$ events are modeled aside, as additional automata. They are not presented here since they are specific to each synchronization mechanism and do not feature any clock nor continuous variable. This task model uses one clock o_i for the periodic activation and one continuous variable x_i which measures the execution time.

Figure 10 gives the model for a fixed priority scheduler. At each event which requires a rescheduling, all the tasks receive a preemption signal; the task with the greatest priority is then started. Priorities are implicit here, and are given by the order in which the scheduler tries to start the tasks. The structure of the scheduler model depends on the number of tasks n and thus the location $Scheduling$ has to be duplicated $n-1$ times. We use one clock u and an invariant $u \leq 0$ to force transitions to be taken without letting time elapse in every location of the scheduler, except in $Idle$.

When possible, we have modeled the periodic activations of tasks with the same period by using only one clock. This however adds one dedicated automaton in the product. The relevance of this step lies in the fact that the verification algorithm complexity is more sensitive to the number of continuous variables than to the number of locations.

Although not presented here, the modeling using *Scheduling*-TPNs, is also generic. Details on this modeling can be found in (Lime and Roux 2003).

5.2 Results

We have compared our method and the direct modeling on a set of systems of increasing complexity: from two processors with four tasks to seven processors

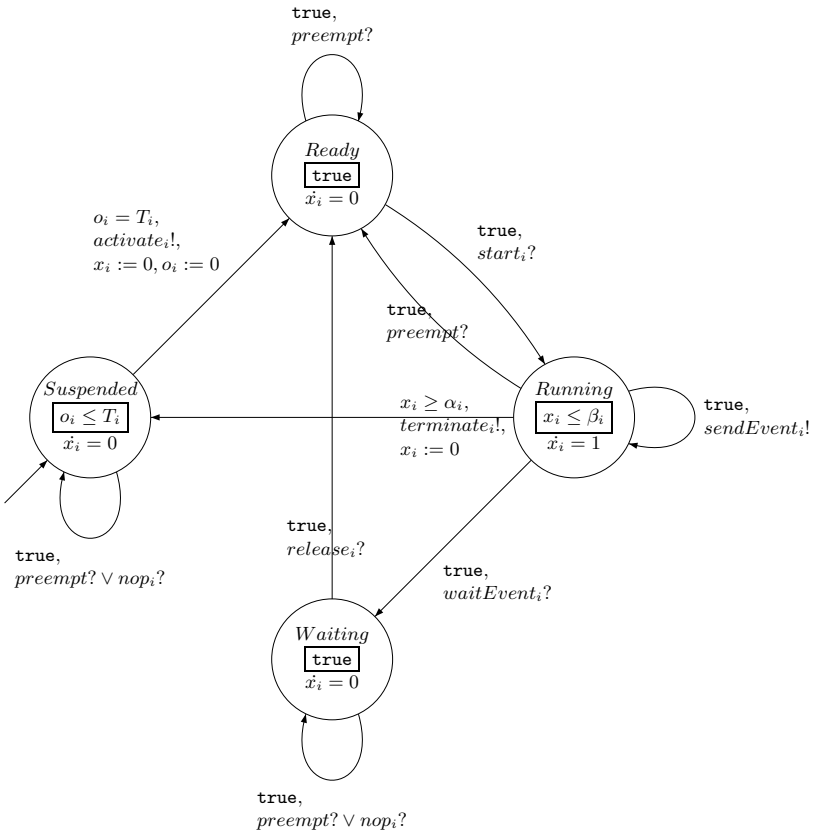

Fig. 9. LHA modeling a real-time task

linked by a CAN bus with eighteen tasks. In these examples, tasks are periodic or launched by periodic tasks (but aperiodic or sporadic tasks can of course be easily treated). There are some precedence constraints modeled by semaphores and some tasks on different processors communicate through the CAN bus. The increasing complexity has been obtained by adding up processors, tasks, dependencies and communications to the first example.

For this kind of models, analytical methods such as presented in (Liu and Layland 1973, Tindell 1994) or computationally simpler formal methods as in (McManis and Varaiya 1994, Fersman *et al.* 2006) are unapplicable or would not give exact results.

Table 1 gives the obtained results.

Columns 2 and 3 give the number of processors and tasks in the system. Columns 4, 5 and 6 describe the direct modeling in HyTech results by giving respectively the number of LHA in the product, the number of continuous variables and the time taken by HyTech to compute the state space. Columns 7, 8,

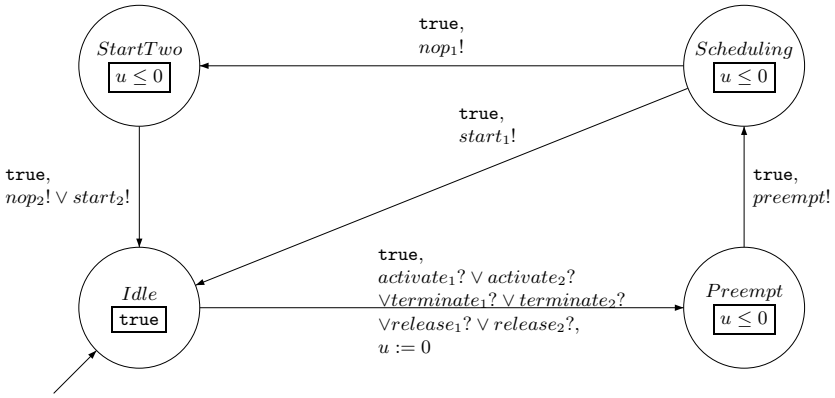

Fig. 10. LHA modeling a Fixed Priority scheduler for two tasks

9 give the results for our method. We give the number of locations, transitions and stopwatches of the LHA generated by our method as well the time taken for its generation. Finally, the last column gives the time used by HyTech to compute the state space of the LHA generated by our method. Times are given is seconds and NA means that the HyTech computation could not give a result on the machine used (a PowerPC G4 1.25GHz with 512MB of RAM).

	Size		Direct modeling (LHA)			Our method (*Scheduling*-TPN $\xrightarrow{\text{Romeo}}$ LHA $\xrightarrow{\text{HyTech}}$ state-space)				
Ex.	Proc.	Tasks	LHA	Var.	HyTech time	Locations	Transitions	Var.	Romeo time	HyTech time
1	2	4	8	7	77.8	20	29	3	$\leq$0.1	0.2
2	3	6	11	9	590.3	40	58	4	$\leq$0.1	0.5
3	3	7	12	10	NA	52	84	4	$\leq$0.1	0.7
4	3+CAN	7	13	11	NA	297	575	7	0.3	5.3
5	4+CAN	9	15	13	NA	761	1677	8	0.9	29.8
6	5+CAN	11	17	15	NA	1141	2626	9	6	60.1
7	5+CAN	12	18	16	NA	2155	5576	9	8.3	56.5
8	6+CAN	14	.	.	NA	4587	12777	10	59.7	438.8
9	6+CAN	15	.	.	NA	4868	13155	11	96.5	1364.3
10	6+CAN	16	.	.	NA	5672	15102	11	439.1	1372.5
11	7+CAN	18	.	.	NA	8817	25874	12	1146.7	NA

Table 1. Experimental results

To compare the efficiency of the two approaches, we can compare the sum of the last two columns to the values in column 6. We see that the computation on a direct modeling as a product of LHA quickly becomes intractable (Example 3). However, with our method, we are able to deal with systems of much greater size, but in the end, the produced automata are too complex for HyTech to analyse. This is the case with the last example: the computation is still possible with

ROMEO but the state space of the resulting LHA is not computable anymore. In this case, if checking safety properties, we can directly exploit the DBM-based extended state-class graph generated by ROMEO by using classical methods such as observers, but keeping in mind that this is an over-approximation.

6 Conclusion

In this paper, we have proposed a method for the verification of timed properties of real-time systems featuring a preemptive scheduling policy including *Fixed Priority* and *Earliest Deadline First*, possibly using *Round-Robin* for tasks with the same priority. The system, modeled as a scheduling time Petri net, is translated into a linear hybrid automaton. We have proved that the initial *Scheduling*-TPN and the obtained LHA are time-bisimilar.

The advantage of this method is manifold: modeling real-time concurrent systems by *Scheduling*-TPNs is quite natural and the resulting state-class LHA can be analyzed and verified using HYTECH, a well-known tool on LHA. Finally the proposed method is efficient for several reasons:

- the additional cost compared to the state-class graph computation is quite low, and the obtained LHA is generally smaller (usually much smaller) than the corresponding state-class graph, so the LHA is often faster to compute than the classical state-class graph;
- this method leads to a single LHA with fewer variables (clocks) than in the initial *Scheduling*-TPN and also than the product of LHA obtained through a generic direct modeling with LHA. As the number of variables of LHA is a critical parameter for the verification efficiency, the ensuing model-checking using HYTECH is much more likely to be tractable;
- the translation can be done by using a fast DBM-based over-approximating method, while still having a resulting LHA that is time-bisimilar to the *Scheduling*-TPN. Then the cost of the translation is fairly lower (almost negligible) than the verification of properties on a direct modeling as a product of LHA.

Practical experimentations show that our method greatly increases the size and complexity of systems for which the state-space can be computed with HYTECH.

Finally, the method may also allow us to specify a real-time system as a mixed model of *Scheduling*-TPN and LHA, and then obtain a LHA modeling the behavior of the whole system.

Acknowledgements.

The authors want to thank Charlotte Seidner for her useful comments on this paper.

References

Altisen, K., G. Gössler, A. Pnueli, J. Sifakis, S. Tripakis and S. Yovine (1999). A framework for scheduler synthesis. In: *20th IEEE Real-Time Systems Symposium (RTSS'99)*. IEEE Computer Society Press. Phoenix, Arizona, USA. pp. 154–163.

Altisen, K., G. Gössler and J. Sifakis (2000). A methodology for the construction of scheduled systems. In: *6th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'00)*. Vol. 1926 of *Lecture Notes in Computer Science*. Springer-Verlag. Pune, India. pp. 106–120.

Altisen, K., G. Gössler and J. Sifakis (2002). Scheduler modelling based on the controller synthesis paradigm. *Journal of Real-Time Systems* **23**, 55–84. Special issue on control-theoritical approaches to real-time computing.

Alur, R. and D.L. Dill (1994). A theory of timed automata. *Theoretical Computer Science* **126**(2), 183–235.

Alur, R., C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis and S. Yovine (1995). The algorithmic analysis of hybrid systems. *Theoretical Computer Science* **138**, 3–34.

Alur, R., T.A. Henzinger and P.-H. Ho (1996). Automatic symbolic verification of embedded systems. *IEEE Transactions on Software Engineering* **22**, 181–201.

Arnold, A. (1994). *Finite Transition System*. Prentice Hall.

Aura, T. and J. Lilius (2000). A causal semantics for time Petri nets. *Theoretical Computer Science*.

Bardin, S., A. Finkel, J. Leroux and L. Petrucci (2003). FAST : Fast acceleration of symbolic transition systems. In: *Proceedings of the 15th International Conference on Computer Aided Verification (CAV'03)*. Vol. 2725 of *Lecture Notes in Computer Science*. Springer-Verlag. pp. 118–121.

Berthomieu, B. and M. Diaz (1991). Modeling and verification of time dependent systems using time Petri nets. *IEEE Transactions on Software Engineering* **17**(3), 259–273.

Berthomieu, B. and M. Menasche (1983). An enumerative approach for analyzing time Petri nets.. *IFIP Congress Series* **9**, 41–46.

Berthomieu, B., D. Lime, O.H. Roux and F. Vernadat (2007). Reachability problems and abstract state spaces for time petri nets with stopwatches. *Journal of Discrete Event Dynamic Systems (jDEDS)* **17**(2), 133–158.

Brémond-Grégoire, Patrice, Insup Lee and Richard Gerber (1993). Acsr: An algebra of communicating shared resources with dense time and priorities. In: *4th International Conference on Concurrency Theory (CONCUR'93)*. Springer-Verlag. London, UK. pp. 417–431.

Bucci, G., A. Fedeli, L. Sassoli and E. Vicario (2003). Modeling flexible real time systems with preemptive time Petri nets. In: *15th Euromicro Conference on Real-Time Systems (ECRTS'2003)*. pp. 279–286.

Bucci, G., A. Fedeli, L. Sassoli and E. Vicario (2004). Time state space analysis of real-time preemptive systems. *IEEE transactions on software engineering* **30**(2), 97–111.

Cassez, F. and K.G. Larsen (2000). The impressive power of stopwatches. In: *11th International Conference on Concurrency Theory, (CONCUR'2000)* (Catuscia Palamidesi, Ed.). number 1877 In: *LNCS*. Springer-Verlag. University Park, P.A., USA. pp. 138–152.

Cassez, F. and O.H. Roux (2006). Structural translation from time petri nets to timed automata – model-checking time petri nets via timed automata. *The journal of Systems and Software* **79**(10), 1456–1468.

Dantzig, G.B. (1963). Linear programming and extensions. *IEICE Transactions on Information and Systems*.

Daws, C. and S. Yovine (1996). Reducing the number of clock variables of timed automata. In: *1996 IEEE Real-Time Systems Symposium (RTSS'96)*. IEEE Computer Society Press. Washington, DC, USA. pp. 73–81.

Dill, D.L. (1989). Timing assumptions and verification of finite-state concurrent systems. In: *Workshop Automatic Verification Methods for Finite-State Systems*. Vol. 407. pp. 197–212.

Fersman, E. and W. Yi (2004). A generic approach to schedulability analysis of real time tasks. *Nordic J. of Computing* **11**(2), 129–147.

Fersman, E., L. Mokrushin, P. Pettersson and W. Yi (2003). Schedulability analysis using two clocks. In: *9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2003)* (Hubert Garavel and John Hatcliff, Eds.). Vol. 2619 of *LNCS*. Springer-Verlag. pp. 224–239.

Fersman, E., L. Mokrushin, P. Pettersson and W. Yi (2006). Schedulability analysis of fixed-priority systems using timed automata. *Theoretical Computer Science* **354**, 301–317.

Fersman, E., P. Petterson and W. Yi (2002). Timed automata with asynchronous processes : Schedulability and decidability. In: *8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'02)*. Vol. 2280 of *LNCS*. Springer-Verlag. Grenoble, France. pp. 67–82.

Gardey, G., D. Lime, M. Magnin and O.H. Roux (2005). Roméo: A tool for analyzing time Petri nets. In: *17th International Conference on Computer Aided Verification (CAV 2005)* (Kousha Etessami and Sriram K. Rajamani, Eds.). Vol. 3576 of *Lecture Notes in Computer Science*. Springer-Verlag. Edinburgh, Scotland, UK. pp. 418–423.

Gardey, G., O.H. Roux and O.F. Roux (2006). 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.

Harbour, M.G., M.H. Klein and J.P. Lehoczky (1991). Fixed priority scheduling of peri-

Lime, D. and O.H. Roux (2006*b*). Vérification formelle des systèmes temps réel avec ordonnancement préemptif. *Techniques et Sciences Informatiques (TSI)* **25**(3), 347–375.

Liu, C. and J.W. Layland (1973). Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of ACM* **20**(1), 44–61.

Magnin, M., D. Lime and O.H. Roux (2005). An efficient method for computing the exact state-space of petri nets with stopwatches. In: *3rd International Workshop on Software Model-Checking (SoftMC 2005)*. Vol. 144 of *Electronic Notes in Theoretical Computer Science*. Elsevier. Edinburgh, Scotland, UK. pp. 59–77.

McManis, J. and P. Varaiya (1994). Suspension automata: A decidable class of hybrid automata. In: *6th International Conference on Computer Aided Verification (CAV'94)* (David L. Dill, Ed.). Vol. 818 of *Lecture Notes in Computer Science*. Springer-Verlag. Stanford, CA, USA. pp. 105–117.

Merlin, P.M. (1974). A Study of the Recoverability of Computing Systems. PhD thesis. Dep. of Information and Computer Science. University of California, Irvine, CA.

Okawa, Y. and T. Yoneda (1996). Schedulability verification of real-time systems with extended time Petri nets. *International Journal of Mini and Microcomputers* **18**(3), 148–156.

OSEK, Group (2001). *OSEK/VDX specification*. http://www.osek-vdx.org.

Palencia, J.C. and M.G. Harbour (1998). Schedulability analysis for tasks with static and dynamic offsets. In: *19th IEEE Real-Time Systems Symposium (RTSS'98)*. IEEE Computer Society Press. Madrid, Spain. pp. 26–37.

Palencia, J.C. and M.G. Harbour (1999). Exploiting precedence relations in the scheduling analysis of distributed real-time systems. In: *20th IEEE Real-Time Systems Symposium (RTSS'99)*. IEEE Computer Society Press. Phoenix, USA. pp. 328–339.

Roux, O. H. and A.-M. Déplanche (2002). A t-time Petri net extension for real time-task scheduling modeling. *European Journal of Automation (JESA)*.

Roux, O.H. and D. Lime (2004). Time Petri nets with inhibitor hyperarcs. Formal semantics and state space computation. In: *The 25th International Conference on Application and Theory of Petri Nets, (ICATPN 2004)* (Jordi Cortadella and Wolfgang Reisig, Eds.). Vol. 3099 of *Lecture Notes in Computer Science*. Springer-Verlag. Bologna, Italy. pp. 371–390.

Tindell, K. (1994). Fixed priority scheduling of hard real-time systems. PhD thesis. Department of Computer Science. University of New York.