

EXPRESSIVENESS AND ANALYSIS OF SCHEDULING EXTENDED TIME PETRI NETS

Didier Lime* Olivier H. Roux*

* IRCCyN, UMR CNRS 6597,
1 rue de la Noë - BP92101
44321 Nantes Cedex 3 - France
{Didier.Lime |
Olivier-h.Roux}@irccyn.ec-nantes.fr

Abstract: The most widely used approach for verifying the schedulability of a real-time system consists of using analytical methods. However, for complex systems with interdependent tasks and variable execution times, they are not well adapted. For those systems, an alternative approach is the formal modelisation of the system and the use of model-checking, which also allows the verification of more varied scheduling properties. In this paper, we show how an extension of time Petri nets proposed in (Roux and Déplanche, 2002), scheduling extended time Petri nets (SETPN), is especially well adapted for the modelisation of real-time systems and particularly embedded systems and we provide a method for computing the state space of SETPN. We first propose an exact computation using a general polyhedron representation of time constraints, then we propose an overapproximation of the polyhedra to allow the use of much more efficient data structures, DBMs. We finally describe a particular type of observers, that gives us a numeric result (instead of boolean) for the computation of tasks response times.

Keywords: Scheduling, formal methods, time Petri nets, realtime systems

1. INTRODUCTION

A real-time system is typically composed of several tasks that interact. An important problem consists of ensuring that the tasks can be executed in such a way that they respect the constraints they are subject to (deadlines, periods, ...), and that the overall system performs correctly with respect to its specification, *i.e.* that it satisfies a given property (for example, the duration between two successive samples of a signal is always in the interval of time $[a, b]$). Solving this problem usually requires scheduling the tasks in an appropriate way.

Two approaches are considered in order to solve the scheduling problem. The off-line approach:

given a system S and a property P , a pre-runtime schedule (scheduler) is constructed in such a way that the system S satisfies P . The on-line approach: a scheduling policy based on priorities, mostly derived from the temporal parameters (e.g. Rate Monotonic, Earliest Deadline, Least Laxity) is selected and implemented within the scheduler ; it is then necessary to make a schedulability analysis to verify that with this given scheduling policy, every task meets its deadline.

Consequently, scheduling theory is studied a lot, mainly in the form of an analytical study. It mostly consists of determining schedulability tests: according to the complexity of the tasks model, these schedulability tests are ex-

acts (necessary and sufficient conditions) or over-approximations (sufficient conditions).

For analysis of sets of independent tasks, exact methods have been developed that take into account both offsets and jitter (Tindell, 1994a; Palencia and Harbour, 1998; Hladik and Déplanche, 2003).

Precedence relations lead to much more complex problems. The main approach tries to fall back into an independent task model by computing an additional jitter simulating the time that the tasks submitted to precedence relations have to wait for ((Tindell, 1994b)). Some works refine this approach by reducing the jitter by a best-case response-time computing ((Henderson *et al.*, 2001) for instance). However, the independent model with jitter is an overapproximation of the original one, which leads to pessimistic response times. Other approaches analyze the precedence graph to reduce the number of scenarios to look at ((Harbour *et al.*, 1991; Palencia and Harbour, 1999)), but they are still based on the identification of a worst-case scenario which does not always exist. Finally, communication between tasks by messages in a distributed system has also been investigated in (Tindell and Clark, 1994), which adds the computation of the messages response times into the classical method.

These works generally consider a fixed execution time but while in independent tasks configurations, the worst case is obtained by considering the longest execution time, this is not true anymore when considering precedence relations, which leads to pessimism in the computation of response times. Moreover, they are not yet well adapted to complex synchronization schemes, such as those allowed by real-time executives services.

Related work

Consequently some work proposes to model complex behaviors of the task using a formal model. Concerning the off-line approach, works are mainly based on the controller synthesis paradigm (Altisen *et al.*, 2000). A scheduler is considered as a controller of the processes to be scheduled, which restricts their behavior by triggering their controllable actions. The models used include Petri nets with Deadlines (Altisen *et al.*, 1999), Petri nets with a maximal firing functioning mode ((Grolleau and Choquet-Geniet, 2002)).

Concerning the on-line approach, the modeling of pre-emptive process scheduling has been the subject of recent works. The first approach consists of modelling priorities of tasks with inhibitor arcs added to the Petri net ((Robert and Juanole,

2000)). An inhibitor arc is then placed from each transition of the pattern representing a task towards each transition of the patterns representing the other lower priority tasks. A similar approach consists of using Petri nets with priorities (Janicki and Koutny, 1999). Okawa and Yoneda ((Okawa and Yoneda, 1996)) propose an approach with time Petri nets consisting of defining groups of transitions together with rates (speeds) of execution. Transition groups correspond to transitions that model concurrent activities and that can be simultaneously ready to be fired. In this case, their rate are then divided by the sum of transition execution rates. Roux and Déplanche (Roux and Déplanche, 2002) propose an extension for time Petri nets (SETPN) that allows to take into account the way the real-time tasks of an application distributed over different processors are scheduled. Finally, Fersman, Pettersson and Yi (Fersman *et al.*, 2002) propose extended timed automata with asynchronous processes. The main idea is to associate each location of a timed automaton with an executable program called a task and to construct the preemptive scheduler (fixed priority or EDF) with timed automata with subtraction. However, they consider the worst case execution times as a fixed time and it is easy to show that reducing the computation time of a task may surprisingly induce a decrease of timing performances for the application.

Except the work of (Fersman *et al.*, 2002), all these models include the concept of stopwatch. The reachability analysis problem for stopwatch automata (as well as for time Petri nets with inhibitor arcs...) is undecidable. There is no guarantee for the termination of the analysis. Moreover, for these models, the state space computation is often inefficient and difficult to implement.

We propose to use the SETPN model (Roux and Déplanche, 2002) that have the advantage of being able to express both concurrency and real-time constraints in a natural way and to express the indeterminism of the task execution time by the firing intervals of transitions. We propose a method for computing the state space and an overapproximation of that method that allows a compact abstract representation of transitions firing domains with DBM (Difference Bound Matrix).

Outline of the paper

The paper is organized as follows : section 2 gives the formal definitions of the SETPN model and of its semantics. Section 3 illustrates the expressivity with regard to the classical services provided by real-time executives. Section 4 describes an exact state space computation and also a more efficient

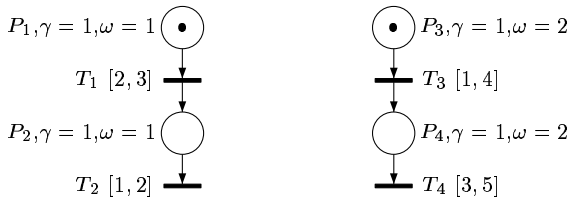


Fig. 1. SETPN of two tasks on one processor

computation method (at the price of an overapproximation) that uses compact abstract representations of the state-space based on DBM to analyze SETPN. Finally, in section 5, we describe special type of observer that allows us to compute response time with our method and we apply this method on a set of examples for which we have exact response times computation methods.

2. SCHEDULING EXTENDED TIME PETRI NETS

This extension of time Petri nets introduced in (Roux and Déplanche, 2002) consists of mapping into the Petri net model the way the different schedulers of the system activate or suspend the tasks. For a fixed priority scheduling policy, SETPN introduce two new attributes (γ and ω) associated to each place that respectively represent allocation (processor) and priority (of the modeled task). From a marking M , a function Act (formally defined in (Roux and Déplanche, 2002)) gives the projection of the behaviour of the different scheduler in the following sense: Let us suppose that the place P models a behavior (or a state) of the task T . $M(p) > 0$ means that the task T is activable and $Act(M(p)) > 0$ means that the task T is active.

All places of a SETPN do not require such parameters. Actually when a place does not represent a true activity for a processor (for example a register or memory state), neither a processor (γ) nor a priority (ω) have to be attached to it. In this specific case ($\gamma = \phi$), the semantics remains unchanged with respect to a standard TPN¹. One can notice that it is equivalent to attach to this place a processor for its exclusive use and any priority. An example of a SETPN is presented in figure 1. The initial marking of the net is $\{P_1, P_3\}$. However, since those two places are affected to the same processor, and that the priority of P_3 is the highest, the initial active marking is $\{P_3\}$. So the first transition fired will be T_3 .

Definition 1. (Scheduling Extended Time Petri net). A scheduling extended time Petri net is a n-tuple $\mathcal{T} = (P, T, \bullet(\cdot), (\cdot)^\bullet, \alpha, \beta, M_0, Act)$, where

- $P = \{p_1, p_2, \dots, p_m\}$ is a non-empty finite set of *places*,
- $T = \{t_1, t_2, \dots, t_n\}$ is a non-empty finite set of *transitions*,
- $\bullet(\cdot) \in (\mathbb{N}^P)^T$ is the *backward incidence function*,
- $(\cdot)^\bullet \in (\mathbb{N}^P)^T$ 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 for each transition respectively its *earliest* and *latest* firing times ($\alpha \leq \beta$),
- $Act \in (\mathbb{N}^P)^P$ is the active marking function. $Act(M)$ is the projection on the marking M of the scheduling strategy. In (Roux and Déplanche, 2002) $Act(M)$ is defined for a fixed priority scheduling policy, starting from three parameters :
 - $Proc\{\phi, proc_1, proc_2, \dots, proc_l\}$ is a finite set of processors (including ϕ that is introduced to specify that a place is not assigned to an effective processor of the hardware architecture),
 - $\omega \in \mathbb{N}^P$ is the priority assignment function,
 - $\gamma \in Proc^P$ is the allocation function.

We define the semantics of scheduling extended time Petri nets as *Timed Transition Systems* (TTS) (Larsen *et al.*, 1995). In this model, two kinds of transitions may occur : *continuous* transitions when time passes and *discrete* transitions when a transition of the net fires.

A *marking* M of the net is an element of \mathbb{N}^P such that $\forall p \in P, M(p)$ is the number of tokens in the place p .

An *active marking* $Act(M)$ of the net is an element of \mathbb{N}^P such that $\forall p \in P, Act(M(p)) = M(p)$ or $Act(M(p)) = 0$.

A transition t is said to be *enabled* by the marking M if $M \geq \bullet t$, (*i.e.* if the number of tokens in M in each input place of t is greater or equal to the valuation on the arc between this place and the transition). We denote it by $t \in enabled(M)$.

A transition t is said to be *active* if it is enabled by the active marking $Act(M)$. We denote it by $t \in enabled(Act(M))$.

A transition t_k is said to be *newly* enabled by the firing of the transition t_i from the marking M , and we denote it by $\uparrow enabled(t_k, M, t_i)$, if the transition is enabled by the new marking $M - \bullet t_i + t_i^\bullet$ but was not by $M - \bullet t_i$, where M is the marking of the net before the firing of t_i . Formally,

$$\uparrow enabled(t_k, M, t_i) = (\bullet t_k \leq M - \bullet t_i + t_i^\bullet) \wedge ((t_k = t_i) \vee (\bullet t_k > M - \bullet t_i))$$

¹ When $\gamma = \phi$, the parameter is omitted in the figures of this paper.

By extension, we will denote by $\uparrow \text{enabled}(M, t_i)$ the set of transitions newly enabled by firing the transition t_i from the marking M .

A *valuation* is a mapping $\nu \in (\mathbb{R}^+)^T$ such that $\forall t \in T, \nu(t)$ is the time elapsed since t was last enabled. Note that $\nu(t)$ is meaningful only if t is an enabled transition. $\bar{0}$ is the *null valuation* such that $\forall k, \bar{0}_k = 0$.

Definition 2. (Semantics of a SETPN). The semantics of a scheduling extended time Petri net \mathcal{T} is defined as a TTS $\mathcal{S}_{\mathcal{T}} = (Q, q_0, \rightarrow)$ such that

- $Q = \mathbb{N}^P \times (\mathbb{R}^+)^T$
- $q_0 = (M_0, \bar{0})$
- $\rightarrow \in Q \times (T \cup \mathbb{R}) \times Q$ is the transition relation including a continuous transition relation and a discrete transition relation.
 - The continuous transition relation is defined $\forall d \in \mathbb{R}^+$ by :

$$(M, \nu) \xrightarrow{d} (M, \nu') \text{ iff } \forall t_i \in T, \nu'(t_i) = \begin{cases} \nu(t_i) & \text{if } \text{Act}(M) < \bullet t_i \\ \wedge M \geq \bullet t_i \\ \nu(t_i) + d & \text{otherwise,} \end{cases}$$

$$M \geq \bullet t_i \Rightarrow \nu'(t_i) \leq \beta(t_i)$$

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

$$(M, \nu) \xrightarrow{t_i} (M', \nu') \text{ iff } \begin{cases} \text{Act}(M) \geq \bullet t_i, \\ M' = M - \bullet t_i + t_i \bullet, \\ \alpha(t_i) \leq \nu(t_i) \leq \beta(t_i), \\ \forall t_k, \nu(t_k)' = \begin{cases} 0 & \text{if } \uparrow \text{enabled}(t_k, M, t_i), \\ \nu(t_k) & \text{otherwise} \end{cases} \end{cases}$$

3. EXPRESSIVENESS

Time Petri nets in general have already been used for a long time to model real-time systems and protocols. SETPN add expressiveness with regard to the scheduling policies provided by real-time executives like OSEK/VDX (*OSEK/VDX specification*, 2001). In this section, we show some examples of modelisation of services of these executives. Modelisation possibilities are of course not restricted to what is presented here.

3.1 Basic task model

The basic model we propose for tasks is shown on Fig. 2a. Basic patterns of that type may be concatenated to express sequentiality (Fig. 2b).

Activation of tasks may be modeled very easily. Fig. 3a shows a periodic activation and Fig. 3b a delayed periodic activation. Cyclic activations are shown on Fig. 3c.

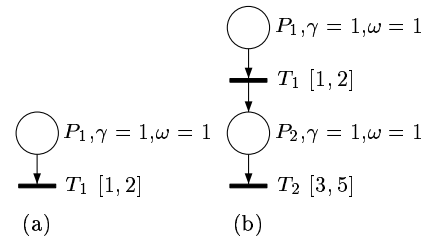


Fig. 2. Basic task model

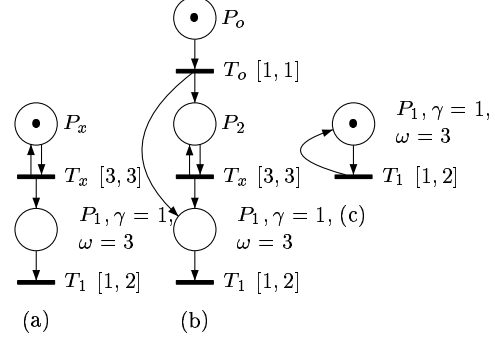


Fig. 3. Activation schemes

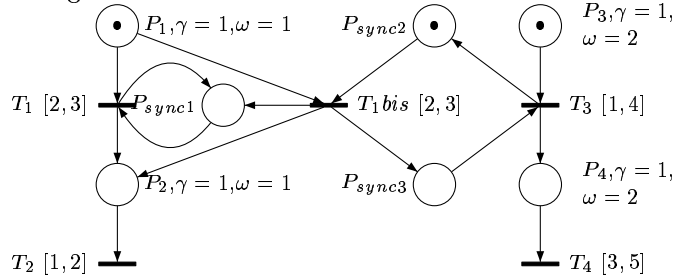


Fig. 4. Synchronization with memorized events

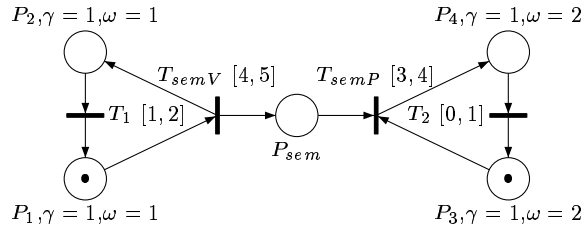


Fig. 5. Two cyclic tasks synchronized *via* a semaphore

3.2 Concurrent tasks

3.2.1. Basic synchronization Real-time executives provide several services for synchronization. In particular, semaphores and events are widely used. Fig. 5 shows a model for Dijkstra's semaphores and Fig. 4 proposes a model for memorized events as found in OSEK/VDX, for instance. Higher level or other synchronization mechanisms are as easily modelisable.

3.2.2. Shared resources access Access to critical resources may involves a mutual exclusion mechanism. This is usually done using a semaphore (Fig. 6). However, it is well-known that this policy may result in deadlocks, so real-time execu-

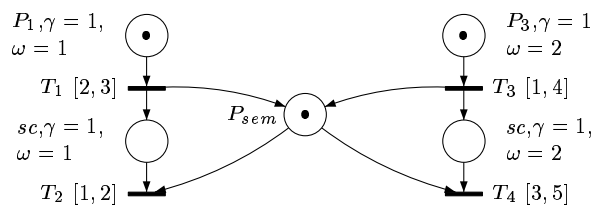


Fig. 6. Semaphore for mutual exclusion

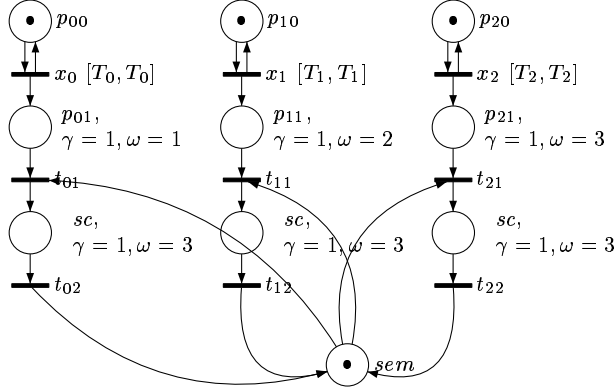


Fig. 7. Priority Ceiling Protocol

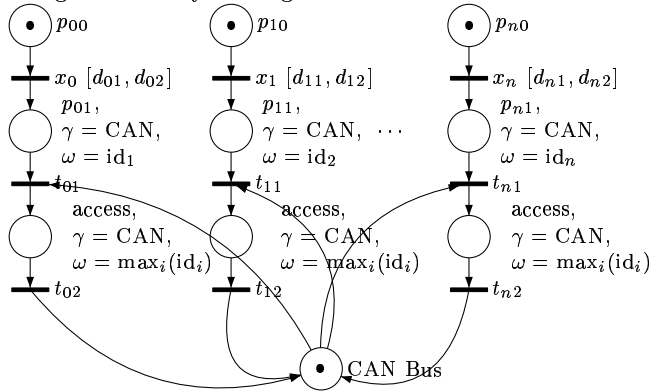


Fig. 8. CAN bus access

tives provide some higher level protocol to handle shared resources access. The most used, in OSEK/VDX ((*OSEK/VDX specification, 2001*)) for instance, is Priority Ceiling Protocol. It can be modelised by SETPN with the pattern in Fig. 7.

3.3 Messaging

We provide here, as an example, a model for the messaging component of a distributed real-time system in which tasks communicate with a CAN bus. This is a high-level modelisation of the CAN protocol, since we only consider that the bus is a shared resource, for which the access is made according to the priorities corresponding to the station number of each node. A much more precise modelisation has been done in (Juanole, 1999), which completely modelize the MAC sublayer of the CAN protocol.

4. ANALYSIS

In order to analyze a time Petri net, the computation of its reachable state space is required.

However, the reachable state space of a time Petri net is obviously infinite, so a method has been proposed to partition it in a finite set of infinite *state classes* (Berthomieu and Diaz, 1991). This method is briefly explained in the next subsection. The following paragraphs then describe its extension in order to compute the state space of SETPN.

4.1 State class graph of a time Petri net

Given a bounded time Petri net, Berthomieu and Diaz have proposed a method for computing the state space as a finite set of *state classes* (Berthomieu and Diaz, 1991). Basically a state class contains all the states of the net between the firing of two consecutive transitions.

Definition 3. (State class). A *state class* C , of a time Petri net, is a pair (M, D) where M is a marking of the net and D a set of inequalities called the *firing domain*.

The inequalities in D are of two types (Berthomieu and Diaz, 1991)

$$\begin{cases} \alpha_i \leq x_i \leq \beta_i \ (\forall i \text{ such that } t_i \text{ is enabled}), \\ -\gamma_{kj} \leq x_j - x_k \leq \gamma_{jk}, \ \forall j, k \text{ such that} \\ j \neq k \text{ and } (t_j, t_k) \in \text{enabled}(M)^2 \end{cases}$$

x_i is the firing time of the enabled transition t_i relatively to the time when the class was entered in.

Because of their particular form, the firing domains may be encoded using DBMs (Dill, 1989), which allow the use of efficient algorithms for the computation of the classes.

Given a class $C = (M, D)$ and a fireable transition t_f , computing the successor class $C' = (M', D')$ obtained by firing t_f is done by:

- (1) Computing the new marking as for classical Petri nets: $M' = M - \bullet t_f + t_f \bullet$
- (2) Making variable substitutions in the domain: $\forall i \neq f, x_i \leftarrow x'_i + x_f$
- (3) Eliminating x_f from the domain using for instance the Fourier-Motzkin method ((Dantzig, 1963))
- (4) Computing a canonical form of the new domain using for instance the Floyd-Warshall algorithm ((Berthomieu, 2001))

With the method for obtaining the successors of a class, computation of the state space of the TPN consists merely of the classical building of the reachability graph of state classes. That is to say, that starting from the initial state class, all the successors obtained by firing fireable transitions are computed iteratively until all the produced successors have already been generated.

4.2 Exact analysis using polyhedra

The semantics of definition 2 obviously implies that the domain of state classes cannot be computed for SETPNs as for classical TPNs. However, minor changes are proposed in (Roux and Déplanche, 2002) to allow its construction.

Precisely, the variable substitution in the firing domain is now only done for *active transitions*. Moreover, when determining firable transitions, the *active firing domain* must be considered *i.e.* the firing domain restricted to variables corresponding to active transitions.

While we still define a state class of a SETPN as a marking and a domain, the general form of the domain is not preserved. The new general form is that of a polyhedron with constraints involving up to n variables, with n being the number of transitions enabled by the marking of the class:

$$\left\{ \begin{array}{l} \alpha_i \leq \theta_i \leq \beta_i, \forall t_i \in \text{enabled}(M), \\ a_{i_1}\theta_{i_1} + \dots + a_{i_n}\theta_{i_n} \leq \gamma_{i_1\dots i_n}, \\ \forall (t_{i_1}, \dots, t_{i_n}) \in \text{enabled}(M)^n \\ \text{and with } (a_{i_0}, \dots, a_{i_n}) \in \mathbb{Z}^n. \end{array} \right.$$

The following paragraphs give the details of the computation of the new domain on a semi-general example. We start from a DBM-like class to show how additional constraints appear.

Let $C = (M, D)$ be a state class of a SETPN, such that

$$D = \left\{ \begin{array}{l} \alpha_1 \leq x_1 \leq \beta_1, \\ \alpha_2 \leq x_2 \leq \beta_2, \\ \alpha_3 \leq x_3 \leq \beta_3, \\ \alpha_4 \leq x_4 \leq \beta_4, \\ -\gamma_{21} \leq x_1 - x_2 \leq \gamma_{12}, \\ -\gamma_{31} \leq x_1 - x_3 \leq \gamma_{13}, \\ -\gamma_{41} \leq x_1 - x_4 \leq \gamma_{14}, \\ -\gamma_{32} \leq x_2 - x_3 \leq \gamma_{23}, \\ -\gamma_{42} \leq x_2 - x_4 \leq \gamma_{24}, \\ -\gamma_{43} \leq x_3 - x_4 \leq \gamma_{34} \end{array} \right. \quad (1)$$

We also suppose that t_1, t_2 and t_3 (corresponding to variables x_1, x_2, x_3) are active, and that t_4 is not. Additionally, we suppose that t_1 is firable and we compute the domain of the class obtained by firing t_1 . The first step is the variable substitution $x_i \leftarrow x'_i + x_1$ for all active transitions but t_1 . The domain becomes

$$\left\{ \begin{array}{l} \alpha_1 \leq x_1 \leq \beta_1, \\ \alpha_2 \leq x_1 + x'_2 \leq \beta_2, \\ \alpha_3 \leq x_1 + x'_3 \leq \beta_3, \\ \alpha_4 \leq x'_4 \leq \beta_4, \\ -\gamma_{21} \leq x_1 - x_1 - x'_2 \leq \gamma_{12}, \\ -\gamma_{31} \leq x_1 - x_1 - x'_3 \leq \gamma_{13}, \\ -\gamma_{41} \leq x_1 - x_4 \leq \gamma_{14}, \\ -\gamma_{32} \leq x_1 + x'_2 - x_1 - x'_3 \leq \gamma_{23}, \\ -\gamma_{42} \leq x_1 + x'_2 - x_4 \leq \gamma_{24}, \\ -\gamma_{43} \leq x_1 + x'_3 - x_4 \leq \gamma_{34} \end{array} \right. \quad (2)$$

The next step is the elimination of the variable x_1 . We use the Fourier-Motzkin method. For that, we rewrite the inequations as follows :

$$\left\{ \begin{array}{l} \alpha_1 \leq x_1, \quad x_1 \leq \beta_1, \\ \alpha_2 - x'_2 \leq x_1, \quad x_1 \leq \beta_2 - x'_2, \\ \alpha_3 - x'_3 \leq x_1, \quad x_1 \leq \beta_3 - x'_3, \\ -\gamma_{41} + x_4 \leq x_1, \quad x_1 \leq \gamma_{14} + x_4, \\ -\gamma_{42} + x_4 - x'_2 \leq x_1, \quad x_1 \leq \gamma_{24} + x_4 - x'_2, \\ -\gamma_{43} + x_4 - x'_3 \leq x_1, \quad x_1 \leq \gamma_{34} + x_4 - x'_3, \\ \alpha_4 \leq x'_4 \leq \beta_4, \\ -\gamma_{32} \leq x'_2 - x'_3 \leq \gamma_{23}, \\ -\gamma_{21} \leq -x'_2 \leq \gamma_{12}, \\ -\gamma_{31} \leq -x'_3 \leq \gamma_{13} \end{array} \right. \quad (3)$$

The Fourier-Motzkin method then consists in writing that the system has solutions if and only if the lower bounds of x_1 are less or equal to the upper bounds. The obtained system is then equivalent to the initial one. After a few simplifications we obtain

$$\left\{ \begin{array}{l} -\gamma_{12} \leq x'_2 \leq \gamma_{21}, \\ -\gamma_{13} \leq x'_3 \leq \gamma_{31}, \\ \alpha_4 \leq x_4 \leq \beta_4, \\ -\gamma_{32} \leq x'_2 - x'_3 \leq \gamma_{23}, \\ -\gamma_{42} - \beta_1 \leq x'_2 - x_4 \leq \gamma_{24} - \alpha_1, \\ -\gamma_{43} - \beta_1 \leq x'_3 - x_4 \leq \gamma_{34} - \alpha_1 \\ \alpha_2 - \gamma_{14} \leq x'_2 + x_4 \leq \beta_2 + \gamma_{41}, \\ \alpha_3 - \gamma_{14} \leq x'_3 + x_4 \leq \beta_3 + \gamma_{41}, \\ \alpha_2 - \gamma_{34} \leq x'_2 + x_4 - x'_3 \leq \beta_2 + \gamma_{43}, \\ \alpha_3 - \gamma_{24} \leq x'_3 + x_4 - x'_2 \leq \beta_3 + \gamma_{42} \end{array} \right. \quad (4)$$

The final step consists in computing a canonical form of this domain and will not be detailed here. What we can see here is that eight inequations, given on the four last lines, are generated, which cannot be expressed with a DBM. Furthermore, we can easily see that those new inequations may give even more complex inequations (*i.e.* involving more variables) when firing another transition for the obtained domain.

4.3 Overapproximation

Manipulating polyhedra in the general case involves a very important computing cost. In order

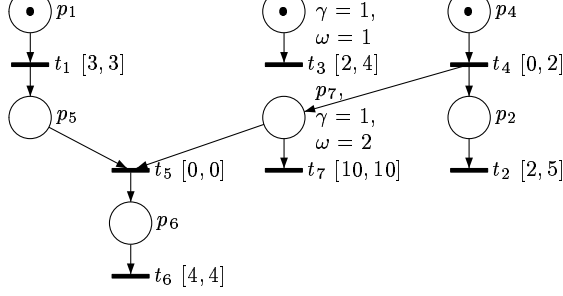


Fig. 9. A Scheduling Extended Time Petri Net

to be able to keep our algorithms efficient for SETPN, we approximate the polyhedron representing the firing domain to the smallest DBM containing it. By doing this, we clearly add states in our classes that should not be reachable and thus we do an overapproximation. This is illustrated by the net in Figure 9. After the firing sequence t_4, t_1, t_5 , the transition t_6 is not fireable because either t_2 or t_3 (depending on the firing time of t_4) must be fired first. The class obtained is:

$$\left\{ \begin{array}{l} \{p_2, p_3, p_6\}, \\ \left\{ \begin{array}{l} 0 \leq x_2 \leq 4, \\ 0 \leq x_3 \leq 4, \\ 4 \leq x_6 \leq 4, \\ -4 \leq x_2 - x_3 \leq 4, \\ -4 \leq x_2 - x_6 \leq 0, \\ -4 \leq x_3 - x_6 \leq 0, \\ 1 \leq x_2 + x_3 \leq 7 \end{array} \right. \end{array} \right.$$

We can easily see that t_6 is indeed not fireable, for this implies $x_2 = x_3 = x_6 = 4$ and thus $x_2 + x_3 = 8$. But if we remove the $x_2 + x_3 \leq 7$ constraint in order to keep a DBM form, t_6 becomes fireable. So we have here an overapproximation.

However, for the verification of safety properties the overapproximation is not a too big concern. Since we want to ensure that something "bad" never happens, we only need to check a set of states which contains the actual state space of the SETPN. Of course, there is still a risk of being pessimistic.

In addition, for the subclass of SETPN with fixed firing times on transitions, the following theorem holds:

Theorem 1. Let $\mathcal{T} = (P, T, \bullet(\cdot), (\cdot)^\bullet, \alpha, \beta, M_0, Act)$ be a SETPN. If $\alpha = \beta$, then inequations in all state classes are reduced to equalities.

Proof. This will be shown by induction: since $\alpha = \beta$, the initial state class domain can be written

$$\left\{ \begin{array}{l} \theta_{i_0} = \alpha(t_{i_0}), \\ \vdots \\ \theta_{i_n} = \alpha(t_{i_n}) \end{array} \right.$$

So the claim of the theorem holds for the initial state class. Let $C = (M, D)$ be a state class such

that

$$D = \left\{ \begin{array}{l} \theta_{i_0} = \alpha_{i_0}, \\ \vdots \\ \theta_{i_n} = \alpha_{i_n} \end{array} \right.$$

Let us suppose that t_{i_n} is a fireable transition from C and $C' = (M, D')$ the class obtained by firing t_{i_n} from C . D' is computed by the four following steps:

- (1) for all active transitions, say t_{i_0}, \dots, t_{i_m} with $m < n$, for instance, the variable substitution $\theta = \theta' + \theta_{i_n}$ is made,
- (2) Disabled transitions (including t_{i_n}) are eliminated from the system of equalities. The resulting system is obviously still composed of equalities,
- (3) Inequalities relative to the newly enabled transitions are added. Since, $\alpha = \beta$, these inequalities are actually equalities,
- (4) The canonical form is not needed since the system of equalities can only be written in one way.

□

As a consequence, we never have overapproximations with that subclass of SETPN. Actually, DBMs are not even required to code that computations.

5. SCHEDULABILITY VERIFICATION

Using the classical observer notion, we can verify varied timed accessibility properties. In order to verify the schedulability, we need a special kind of observers that we formally describe in the following section. While classical observers (Toussaint *et al.*, 1997) give a boolean response for a given property, thanks to these observers we are also able to compute the response time of the tasks that we have modelised and then to compare with other approaches.

5.1 Response time observers

Definition 4. (observer). Let a scheduling extended time Petri net

$\mathcal{T} = (P, T, \bullet(\cdot), (\cdot)^\bullet, \alpha, \beta, M_0, Act)$, and two transitions $\{in, out\} \in T$.

$\mathcal{T}_{\mathcal{O}_{in,out}} = (P_o, T, \bullet_o(\cdot), (\cdot)^\bullet_o, \alpha, \beta, M_{o0})$ is the TPN \mathcal{T} observed by the observer

$\mathcal{O}_{in,out}(\mathcal{T}) = (p_{obs}, x, m_{obs})$ where

- $P_o = P \cup \{p_{obs}\}$,
- $M_0(p_{obs}) = m_{obs}$,
- $\bullet_o out = \bullet out + p_{obs}$ and $\forall t \in T - \{out\}$, $\bullet_o t = \bullet t$,
- $in^\bullet_o = in^\bullet + p_{obs}$ and $\forall t \in T - \{in\}$, $t^\bullet_o = t^\bullet$,

- $\forall M \in \mathbb{N}^P$, $Act_o(M(p_{obs})) = M(p_{obs})$ and $\forall p \in P$, $Act_o(M(p)) = Act(M(p))$.

We note \mathcal{Q} the set of state $Q = (M, \nu, \nu_x)$ of the observed TPN $\mathcal{T}_{\mathcal{O}_{in,out}}$. The semantics of an observed time Petri net $\mathcal{T}_{\mathcal{O}_{in,out}}$ is the semantics of the TPN \mathcal{T} with the following modification : The continuous transition relation is defined $\forall d \in \mathbb{R}^+$ by :

$$(M, \nu, \nu_x) \xrightarrow{d} (M, \nu', \nu'_x) \text{ iff } \left\{ \begin{array}{l} \forall t_i \in T \left\{ \begin{array}{l} \nu'(t_i) = \begin{cases} \nu(t_i) & \text{if } Act(M) < \bullet t_i \\ \wedge M \geq \bullet t_i \\ \nu(t_i) + d & \text{otherwise} \end{cases} \\ M \geq \bullet t_i \Rightarrow \nu'(t_i) \leq \beta(t_i) \end{array} \right. \\ M(p_{obs}) \geq 1 \Rightarrow \nu'_x = \nu_x + d \\ M(p_{obs}) = 0 \Rightarrow \nu'_x = \nu_x = 0 \end{array} \right.$$

As the transition *in* models the request of a task t_a and the transition *out* its termination, we can use this observer to determine the response time of t_a and to check that the task respects its deadline.

For a schedulable task such that $M(p_{obs}) \leq 1$, the response time of a task *task* is :

$$t_r(task) = \max_{\mathcal{Q}}(\nu_x)$$

One can notice that since inequations in the domain cannot be strict, a task for which a null execution time remains may be preempted while it has not done the discrete transition corresponding to its termination.

5.2 Experimental results

For testing, we have compared the results of the computation of response times given by a modelisation with HyTech (Henzinger *et al.*, 1997) for an exact result on relatively small but complex models, analytical methods for bigger but simpler models (independent tasks) and our method using the DBM overapproximation.

Experimentation on several hundreds of examples of varied models of real-time systems with non-fixed execution times gives the same results for the three (or two when HyTech or analytical method is not adapted) methods. This allows us to think that there is a relatively big subclass of SETPN for which the DBM algorithm is exact, or at least that the approximation is quite small on classical real-time applications.

6. CONCLUSION

In this paper, we have shown how to express most of the features of real-time systems with SETPN, from basic activation schemes for tasks

to complex resource access protocols. We also provide a theoretical framework for analysis of SETPN. In particular, we have given an exact method for computing the state space of a SETPN as well as a faster method at the cost of an overapproximation. While this approximation is not quantified, experimental results allow us to think it is not very important. We have also given a formal description of a special type of observers that allow us to compute response times of tasks modeled by SETPN.

Further work includes the identification of the class of SETPN for which there is no overapproximation, extension of the modelisation for the round-robin and dynamic scheduling policies and the generation of the state-space as timed automaton.

REFERENCES

- Altisen, K., G. Goßler, 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. Goßler 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.
- Berthomieu, B. (2001). La méthode des classes d'états pour l'analyse des réseaux temporels. In: *3e congrès Modélisation des Systèmes Réactifs (MSR'2001)*. Hermes. Toulouse, France. pp. 275–290.
- 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.
- Dantzig, G. B. (1963). Linear programming and extensions. *IEICE Transactions on Information and Systems*.
- 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., 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 *Lecture Notes in Computer Science*. Springer-Verlag. Grenoble, France. pp. 67–82.

- Grolleau, Emmanuel and Annie Choquet-Geniet (2002). Off-line computation of real-time schedules using petri nets. *Discrete Event Dynamic Systems* **12**(3), 311–333.
- Harbour, M.G., M.H. Klein and J.P. Lehoczky (1991). Fixed priority scheduling of periodic tasks with varying execution priority. In: *12th IEEE Real-Time Systems Symposium (RTSS'91)*. IEEE Computer Society Press. San Antonio, USA. pp. 116–128.
- Henderson, W., D. Kendall and A. Robson (2001). Improving the accuracy of scheduling analysis applied to distributed systems. *Real-Time Systems* **20**(1), 5–25.
- Henzinger, T.A., P.-H. Ho and H. Wong-Toi (1997). Hytech: A model checker for hybrid systems. *Journal of Software Tools for Technology Transfer* **1**(1-2), 110–122.
- Hladik, P.-E. and A.-M. Déplanche (2003). Analyse d'ordonnancement de tâches temps-réel avec offset et gigue. In: *11th international Conference on Real-Time Systems (RTS'03)*. Paris, France. p. to appear.
- Janicki, R. and M. Koutny (1999). On causality semantics of nets with priorities. *Fundamenta Informaticae* **38**(3), 223–255.
- Juanole, G. (1999). Modélisation et évaluation du protocole mac du réseau can. In: *Rapport LAAS No99303. Ecole d'Eté Applications, Réseaux et Systèmes (ETR'99)*. Poitiers, France. pp. 187–200.
- Larsen, K. G., P. Pettersson and W. Yi (1995). Model-checking for real-time systems. In: *Fundamentals of Computation Theory*. pp. 62–88.
- 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/VDX specification* (2001). <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, Arizona, USA. pp. 328–339.
- Robert, P.H. and G. Juanole (2000). Modélisation et vérification de politiques d'ordonnement de tâches temps-réel. In: *8th Colloque Francophone sur l'Ingénierie des Protocoles (CFIP'00)*. Hermes. Toulouse, France. pp. 167–182.
- 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)*.
- Tindell, K. (1994a). Adding time-offsets to schedulability analysis. Technical Report YCS-94-221. University of York, Computer Science Department.
- Tindell, K. (1994b). Fixed priority scheduling of hard real-time systems. PhD thesis. Department of Computer Science. University of New York.
- Tindell, K. and J. Clark (1994). Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and Microprogramming* **40**(1-2), 117–134.
- Toussaint, J., F. Simonot-Lion and Jean-Pierre Thomesse (1997). Time constraint verifications methods based time petri nets. In: *6th Workshop on Future Trends in Distributed Computing Systems (FTDCS'97)*. Tunis, Tunisia. pp. 262–267.