A Translation Based Method for the Timed Analysis of Scheduling Extended Time Petri Nets

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 present a method for the timed analysis of real-time systems, taking into account the scheduling constraints. The model considered is an extension of time Petri nets, Scheduling Extended Time Petri Nets (SETPN) for which the valuations of transitions may be stopped and resumed, thus allowing the modelling of preemption. This model has a great expressivity and allows a very natural modelling. The method we propose consists of precomputing, with a fast algorithm, the state space of the SETPN as a stopwatch automaton (SWA). This stopwatch automaton is proven timed bisimilar to the SETPN, so we can perform the timed analysis of the SETPN through it with the tool on linear hybrid automata, HYTECH. *The main interests of this pre-computation are that it is fast because it is Difference Bounds Matrix (DBM)-based, and that it has online stopwatch reduction mechanisms. Consequently, the resulting stopwatch automaton has, in the general case, a fairly lower number of stopwatches than what could be obtained by a direct modelling of the system as SWA. Since the number of stopwatches is critical for the complexity of the verification, the method increases the efficiency of the timed analysis of the system, and in some cases may just make it possible at all.*

1. Introduction

Hard real-time systems are becoming more and more complex and are often critical. Therefore thorough verification of such systems has to be performed, including behaviour and timing correctness. These systems are usually designed as several tasks interacting and sharing one or more processors. Hence, in a system S, tasks have to 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 as S satisfies P_i. In the online approach, scheduling of the tasks is done at runtime according to a scheduling policy based on priorities (e.g. Rate Monotonic, Earliest Deadline). Then we have to verify that S satisfies the properties P_i, the first one of which is schedulability, that is that each task meets its deadline. This is the approach taken in this paper.

Analytical online scheduling analysis

The online scheduling analysis has been much studied and many analytical results have been proposed concerning mostly the schedulability of task sets since Liu and Layland in 1973 [16]. Low cost exact analysis of sets of independent tasks with fixed execution times are available in [23, 19, 11] for instance. Extensions have been proposed to take into account interactions between tasks and variable execution time, *e.g.* [20, 9]. They 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) [1] and timed Petri nets (TPN) [17].

Formal models for online scheduling

Some papers consider the worst case execution times of a task as a fixed time and then they can use models such as timed automata with subtraction [8]. However it is easy to show that, in the context of inter-dependent tasks, reducing the computation time of a task may surprisingly induce a decrease of timing performances for the application. In the general case, to be able to model the execution times of the tasks as well as preemptions, a timed model which is able to express intervals of time, with the concept of stopwatch (a clock that can be stopped and resumed) is required. In this

class of model, one can find stopwatch extensions of classical dense time model: timed automata (TA) and time Petri nets (TPN). Stopwatch automata (SWA) [5] extend timed automata with stopwatches. They allow the modelling of real-time tasks as well as the behavior of a scheduler. The model of a real-time system is then obtained as the synchronized product of these SWA. Cassez and Larsen [5] prove that stopwatch automata with unobservable delays are as expressive as linear hybrid automata (LHA) in the sense that any timed language acceptable by a LHA is also acceptable by a SWA. The reachability problem is undecidable for LHA and also undecidable for SWA.

Several papers are also interested in extending time Petri nets. Okawa and Yoneda ([18]) 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. Finally, Roux and Déplanche [21] 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. The same approach is developed in [4, 3], with preemptive time Petri nets. These last two models are subclasses [1] of inhibitor hyperarcs time Petri nets (IHTPN) [22], which, since they have a more general purpose, are not as well-suited for modelling real-time systems. For these four models, as for time Petri nets, boundedness and reachability are undecidable. Boundedness and other results are then obtained by computing the state space if it is computable.

State-space computation

For dense-time models, the state space is generally infinite, because of the real-valued clocks, so, we need to group some states together, in order to obtain a finite number of these groups, which is hopefully computable. These groups of states are, for instance, *regions* and *zones* for timed automata or *state classes* for time Petri nets. If the model does not have 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 Difference Bound Matrix (DBM). DBM allow fast manipulation and generation (*i.e.* polynomial complexity). When the model has stopwatches, inequations describing the group of states are more complex and do not fit into a DBM anymore. 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 ([5, 14, 4]). This is clearly an over-approximation.

In [4], the authors also propose an interesting method for quantitative timed analysis. As just discussed, they over-approximate the computation of the state class graph by using Difference Bounds Matrix. Then, given an untimed transition sequence from the over-approximated state class graph, they can obtain the feasable timings between the firing of the transitions of the sequence as the solution of a linear programming problem. In particular, if there is no solution, the transition sequence has been introduced by the over-approximation and can be cleaned up, otherwise the solution set allows to check timed properties on the firing times of transitions.

Number of clocks

The number of clocks/stopwatches 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 becomes quickly intractable with the tool on linear hybrid automata HYTECH ([10]). Algorithms exist for timed automata, such as [7], to reduce the number of clocks. As far as we know, there are no such algorithms for hybrid automata. Anyway, these algorithms can only be applied to single automata, not products, and convenient modelling of real-time systems using SWA can only be achieved through products.

Our contribution

In this paper, we consider the SETPN model, for its great expressively and its fitness to the scheduling problem [14]. For this model, we tackle the problem of the state space explosion by a two-stage analysis. First we pre-compute the state space of the SETPN as a stopwatch automaton. This first step is performed by a fast DBM-based algorithm. While this algorithm is over-approximating, the produced stopwatch automaton is proved to be timed bisimilar to the initial SETPN *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 SWA with the HYTECH model-checker. For this second step to be efficient, the number of stopwatches must be as low as possible. To this effect, the translation algorithm offers stopwatch reduction mechanisms and thus yields a stopwatch automaton that has, in the general case, a fairly lower number of stop-

1 Whether all these models actually define the same class of nets or not is an open problem.

watches than what is required for a direct modelling as a product of stopwatch automata.

The rest of the paper is organized as follows: section 2 presents the SETPN model and a short description of the state space computation for SETPN, section 3 describes the translation into a stopwatch automaton and proves the correctness of the translation with a bisimulation. Finally, we give a case study in section 4.

2. Scheduling extended time Petri nets

This extension of time Petri nets introduced in [21] consists of mapping into the Petri net model the way the different schedulers of the system activate or suspend the tasks. Scheduling extended time Petri nets (SETPN) are well adapted to the modelling of concurrent systems. For instance, it is very easy to model semaphores *independently* of the number of tasks which may be blocking on them with SETPN, while it is not possible with timed automata (without extra variables). Furthermore, the behavior of the scheduler is included in the semantics of the SETPN model. This has the advantage that the designer does not have to worry about modelling the scheduler. In addition, it generally slightly decreases the complexity of the model, in terms of size and number of clocks/stopwatches.

From a marking M, a function Act (formally defined in [21]) gives the projection of the behavior of the different schedulers 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 ready and $Act(M(p)) > 0$ means that the task T is active.

For the particular case of fixed priority scheduling policies, we introduce two new attributes (γ and ω) associated to each place of the SETPN that respectively represent allocation (processor) and priority (of the modeled task). 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 TPN [2]. 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 an 8-tuple* $\mathcal{T} = (P, T, {}^\bullet(.), (.)^\bullet, \alpha, \beta, M_0, Act)$, *where*

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

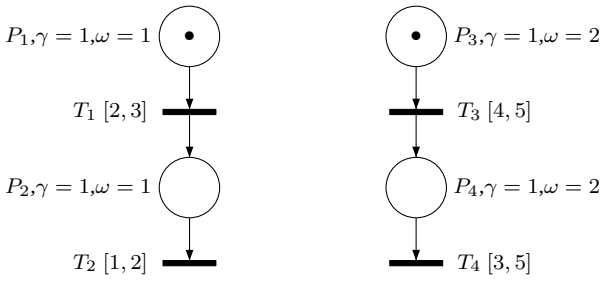

Figure 1. SETPN of two tasks on one processor

- $P = \{p_1, p_2, \ldots, p_m\}$ *is a non-empty finite set of* places,
- $T = \{t_1, t_2, \ldots, t_n\}$ *is a non-empty finite set of* transitions ,
- ${}^\bullet(.) \in (\mathbb{N}^P)^T$ *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 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 [21]* $Act(M)$ *is defined for a fixed priority scheduling policy, starting from three parameters :*
  - $Proc\{\phi, proc_1, proc_2, \ldots, proc_l\}$ *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) [12]. 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,

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

By extension, we will denote by $\uparrow enabled(M, t_i)$ 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. $\overline{0}$ is the *null valuation* such that $\forall k, \overline{0}_k = 0$.

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*

- $Q = \mathbb{N}^P \times (\mathbb{R}^+)^T$
- $q_0 = (M_0, \overline{0})$
- $\rightarrow \in Q \times (T \cup \mathbb{R}) \times Q$ *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:*

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

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

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

2.1. State class graph of a SETPN

In [14], we have given a semi-algorithm for computing the state space of a SETPN. This methods is an extension of the state class graph method of Berthomieu and Diaz on time Petri nets [2].

State classes are still defined as a pair with a *marking* and a *firing domain*. However, with the presence of stopwatches (here the valuation of the clocks), the firing domain of state classes cannot be encoded into a Difference Bound Matrix (DBM) anymore ; a general polyhedron form is required.

As a consequence, we need a new definition for the firability of a transition from a class:

Definition 3 (Firability) *Let $C = (M, D)$ be a state class of a SETPN. A transition t_i is said to be* firable *from C iff there exists a solution $(\theta_0^*, \ldots, \theta_{n-1}^*)$ of D, such that $\forall j \in [\![0, n-1]\!] - \{i\}$, s.t. t_j is active , $\theta_i^* \leq \theta_j^*$.*

Now, given a class $C = (M, D)$ and a firable transition t_f, the class $C' = (M', D')$ obtained from C by the firing of t_f is given by

- $M' = M - {}^\bullet t_f + t_f{}^\bullet$
- D' is computed along the following steps, and noted $next(D, t_f)$
  1. variable substitutions for all enabled transitions that are *active* t_j: $\theta_j = \theta_f + \theta'_j$,
  2. intersection with the set of positive or null reals $\mathbb{R}^+$: $\forall i, \theta'_i \geq 0$,
  3. elimination (using for instance the Fourier-Motzkin method [6]) of all variables relative to transitions disabled by the firing of t_f,
  4. addition of inequations relative to newly enabled transitions

$$\forall t_k \in \uparrow enabled(M, t_f), \alpha(t_k) \leq \theta'_k \leq \beta(t_k).$$

The state class graph of a SETPN is then defined as the quotient of the infinite graph generated by computing iteratively all the successors by $next$ of the initial class by the equivalence relation of definition 4, in which $[\![D]\!]$ denotes the set of solutions of the inequation set D.

Definition 4 (Equality of state classes) *Two classes $C_1 = (M_1, D_1)$ and $C_2 = (M_2, D_2)$ are* equal *if $M_1 = M_2$ and $[\![D_1]\!] = [\![D_2]\!]$.*

2.2. DBM over-approximation

As shown in [14], we can speed up the computation of the state class graph of a SETPN by approximating the domain of each generated class to a DBM containing it. The obvious consequence is that we add states to the computed state space that are not reachable. In particular, in some cases, this could prevent the computation to terminate, by making the number of computed markings unbounded. Conversely, this can also make the computation terminate by cutting off some of the constraints preventing the convergence (for instance, some nets have successive domains

such that when firing n times the same sequence of transitions, some inequations in the obtained domain are of the form $n\theta \leq 2n+1$).

3. State class stopwatch automaton

In this section, we present a method for computing the state space of a SETPN as a stopwatch automaton. We prove the soundness of the computation by proving that this SWA is timed bisimilar to the initial SETPN. Then we show how to obtain, much faster with a DBM-based over-approximating method, a SWA which is also timed bisimilar to the SETPN. But first, we define stopwatch automata.

3.1. Stopwatch automata

We basically define stopwatch automata (SWA) as timed automata with stopwatches:

Definition 5 (Stopwatch Automaton) *A* Stopwatch 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 positive real-valued stopwatches,*
- A *is a finite set of* actions,
- $E \subset L \times \mathcal{C}(X) \times A \times 2^X \times L$ *is a finite set of edges. If* $e = (l, \delta, \alpha, R, \rho, l') \in E$. e *is the edge between the locations* l *and* l'*, with the* guard δ*, the* action α*, the set of stopwatches to* reset R *and the* clock assignment *function* ρ.
- $Inv \in \mathcal{C}(X)^L$ *maps an* invariant *to each location*
- $Dif \in (\{0,1\}^X)^L$ *maps an* activity *to each location,* $\dot{X}$ *being the set of derivatives of the stopwatches w.r.t. time, that is to say their changing rates.* $\dot{X} = (Dif(l)x)_{x \in X}$.

For short, given a location l, a stopwatch x and $b \in \{0,1\}$, we will denote $Dif(l)(x) = b$ by $\dot{x} = b$ when the location considered is not ambiguous.

Definition 6 (Semantics of a SWA) *The semantics of a SWA* 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*
$$\begin{cases} \nu' = \nu + \dot{X} * t, \\ \forall t' \in [0,t], Inv(l)(\nu + \dot{X} * t') = \texttt{true} \end{cases}$$

3.2. State class stopwatch automaton

Following the idea of [15] on classical time Petri nets, we extend the notion of state classes with information about the stopwatches 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 stopwatch automaton from the extended state class graph.

So 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 *is a firing domain,* χ *is a set of stopwatches and* $trans \in (2^T)^\chi$ *maps stopwatches to sets of transitions.*

The stopwatches in χ measure the cumulated time during which the transitions associated 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:

1. M' and D' are computed as in section 2,
2. for each stopwatch x in χ, the disabled transitions are removed from $trans(x)$,
3. the stopwatches whose image by $trans$ is empty are removed from χ,
4. if there are newly enabled transitions by the firing of t_f, two cases can occur:
   - there exists a stopwatch x whose value is 0. Then, we simply add the newly enabled transitions to $trans(x)$,
   - such a stopwatch does not exist. Then we need to create a new stopwatch x_i associated to the newly enabled transitions. The index, i, is chosen as the smallest available index among the stopwatch of χ. We add x_i to χ and $trans(x_i)$ is the set of newly enabled transitions
5. if all the transitions associated to a stopwatch are inactive (resp. active) that stopwatch is stopped (resp. resumed),
6. if the image by $trans$ of a running (resp. stopped) stopwatch x contains both active and inactive transitions, then a new stopwatch x' is created as for newly enabled transitions, to which are associated by $trans$ the inactive (resp. active) transitions. That stopwatch is stopped (resp. started) and its value is set to that of x: $x' = x$.

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 convergence criterion is *stopwatch-similarity*:

Definition 8 (stopwatch-similarity) *Two extended state classes* $C = (M, D, \chi, trans)$ *and* $C' = (M', D', \chi', trans')$ *are* stopwatch-similar, *and we denote it by* $C \approx C'$, *iff they have the same markings, the same number of stopwatches and their stopwatches are mapped to the same transitions:*

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

We can easily see that if two classes are stopwatch-similar, the stopwatches that are associated to the same transitions have the same activity (running or suspended), so the definition is coherent.

So, when two classes are stopwatch-similar, if we also have an inclusion according to definition 9, then we stop the exploration of the current branch. If we do not, we make loop anyway but continue the computation of the successors of the states that are not in the intersection of the two domains.

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 stopwatch-similar and* $[\![D']\!] \subset [\![D]\!]$. *This is denoted by* $C' \subset C$.

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

- $C^{ext} = \mathbb{N}^P \times \mathbb{R}^T \times 2^X \times (2^T)^X$, X being the set of all stopwatches,
- $C_0 = (M_0, D_0, \chi_0, trans_0)$, where M_0 is the initial marking, $D_0 = \{\alpha(t_i) \leq \theta_i \leq \beta(t_i) | \, t_i \in enabled(M_0)\}$, $\chi_0 = \{x_0, x_1\}$ and $trans_0 = ((x_0, enabled(Act(M_0))), (x_1, enabled(M_0) - enabled(Act(M_0))))$,
- $\rightarrow^{ext} \in C^{ext} \times T \times C^{ext}$ is the transition relation defined by the above rules.

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

Definition 10 (State Class Stopwatch Automaton) *The state class stopwatch 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 stopwatches*

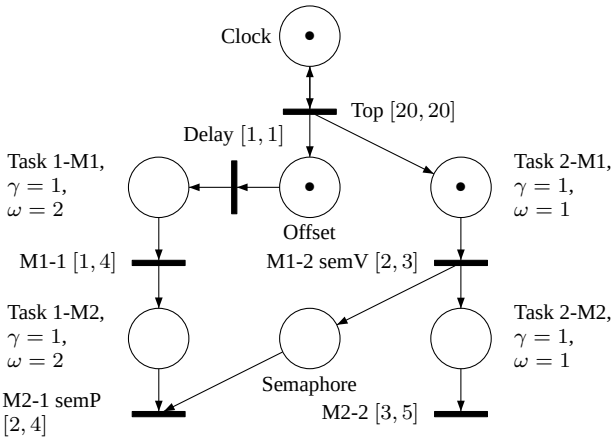

Figure 2. SETPN of two tasks on one processor with a semaphore

- $A = T$ *is the set of transitions*
- E *is the set of edges defined as follows,*

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

- $\forall C \in C^{ext}, Inv(C) = \bigwedge_{x \in \chi, t \in trans(x)} (x \leq \beta(t))$.
- $\forall C \in C^{ext}, \forall x \in \chi, Dif(C)(x) = 1$ *if* $\forall t \in trans(x), t$ *is active*, $Dif(C)(x) = 0$, *otherwise.*

As an example, Figure 2 shows a SETPN modelling two periodic tasks running on the same processor and synchronized by a semaphore. Figure 3 shows the corresponding state class SWA.

3.3. Termination of the algorithm

As for time Petri nets, reachability is undecidable for SETPN (and for SWA). For bounded TPN and timed automata, it is decidable. However, according to some of our most recent work, reachability is also undecidable for bounded SETPN. As a consequence, the computation of the state class stopwatch automaton is not guaranteed to finish, which is inherent to that class of models.

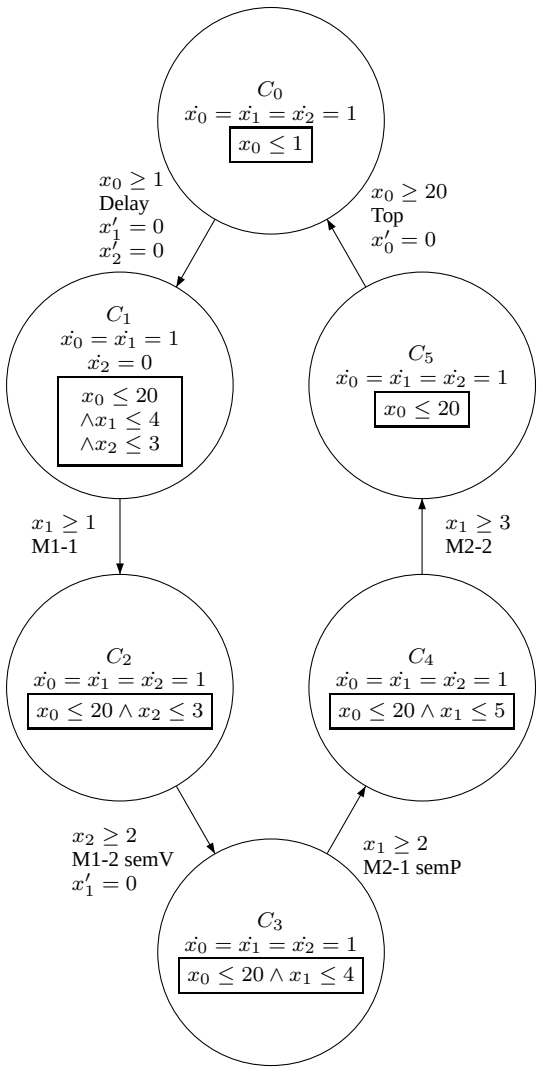

Figure 3. A state class stopwatch automaton. Initial location is C_0.

3.4. Soundness of the translation

In order to prove the soundness of this expression of the state space of a SETPN, we will show in theorem 1 that the SETPN and its state class SWA are timed bisimilar.

Theorem 1 (Bisimulation) *Let $Q_{\mathcal{T}}$ be the set of states of the SETPN $\mathcal{T}$ and $Q_{\mathcal{A}}$ the set of states of the state class stopwatch 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 $\forall s = (M_{\mathcal{T}}, \nu_{\mathcal{T}}) \in Q_{\mathcal{T}}, \forall a = (l, \nu_{\mathcal{A}}) \in Q_{\mathcal{A}}, s\mathcal{R}a \Leftrightarrow M_{\mathcal{T}} = M_{\mathcal{A}}$ if $M_{\mathcal{A}}$ is the marking of the extended state class l and $\forall t \in enabled(M_{\mathcal{T}}), \exists x_t \in X, \nu_{\mathcal{T}}(t) = \nu_{\mathcal{A}}(x_t)$ and $\dot{x} = 1$ if t is active and $\dot{x} = 0$ otherwise.*

$\mathcal{R}$ is a bisimulation.

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

1. Let us suppose that the SETPN can let the time $\tau \in \mathbb{R}^+$ elapse: $s \xrightarrow{\tau} s'$. That means that $\forall t \in enabled(M_{\mathcal{T}})$, $\nu_{\mathcal{T}}(t) + \tau \leq \beta(t)$. So, $\nu_{\mathcal{A}}(x_t) + \tau \leq \beta(t)$ and so by definition of $Inv(l)$, $Inv(l)(\nu_{\mathcal{A}} + \tau')$ is true $\forall \tau' \leq \tau$. And therefore, the SWA $\mathcal{A}$ can let the time τ elapse: $a \xrightarrow{\tau} a'$. Since the SETPN stays in the same marking, and the SWA in the same location, the activity of transitions and the conditions on $\dot{x}$ do not change. As a consequence, $\nu_{\mathcal{T}} + \tau = \nu_{\mathcal{A}} + \tau$ and finally $s'\mathcal{R}a'$.

2. Let us suppose that the SETPN can fire the transition $t \in T$: $s \xrightarrow{t} s'$. By definition of the state class stopwatch 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 and therefore, the SWA $\mathcal{A}$ can take the edge e: $a \xrightarrow{t} a'$. By definition of l', the marking $M'_{\mathcal{A}}$ of the extended class l' is the same as the new marking $M'_{\mathcal{T}}$ of the SETPN. Let t' be a transition in $enabled(M'_{\mathcal{T}})$. If t' is newly enabled, a new stopwatch is created or it is associated to a stopwatch whose value is 0 and whose state (running or stopped) is the same as that of t'. In the first case, the state of the stopwatch is also set accordingly to the activity of t'. If t' is not newly enabled, and if all transitions associated to its stopwatch have the same activity, then the state of the stopwatch is set accordingly to the activity of t', else, and if all transitions associated to that stopwatch have not the same activity, a new stopwatch is created to which is associated t' with the appropriate state. In short, $s'\mathcal{R}a'$ by construction.

3. Let us suppose that the SWA can let the time $\tau \in \mathbb{R}^+$ elapse: $a \xrightarrow{\tau} a'$. That means that $Inv(l)(\nu_{\mathcal{A}} + \tau)$ is true. So, by definition of $Inv(l)$, $\forall x \in X, \forall t \in trans(x)$, $\nu_{\mathcal{A}}(x) + \tau \leq \beta(t)$, which is equivalent to $\forall x \in X, \forall t \in trans(x)$, $\nu_{\mathcal{T}}(t) + \tau \leq \beta(t)$. Since $\bigcup_{x \in X} trans(x) = enabled(M_{\mathcal{T}})$, we have finally, $\forall t \in enabled(M_{\mathcal{T}}), \nu_{\mathcal{T}}(t) + \tau \leq \beta(t)$, which means that $\mathcal{T}$ can let the time τ elapse: $s \xrightarrow{\tau} s'$. As for the first point, $\nu_{\mathcal{T}} + \tau = \nu_{\mathcal{A}} + \tau$ and so, $s'\mathcal{R}a'$.

4. Let us suppose that the SWA 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 so $\nu_{\mathcal{T}}(t) \geq \alpha(t)$, which means that t is firable for $\mathcal{T}$: $s \xrightarrow{t} s'$. Like in the second point, $s'\mathcal{R}a'$ by construction.

□

3.5. Number of stopwatches

As mentioned before, the number of stopwatch is a critical concern for the computation of the state space of formal models. So, in this method we take great care so as to keep the number of stopwatches as low as possible. modelling with SETPN requires roughly the same number of stopwatches [3] as a direct modelling as a product of SWA (minus the possible stopwatches of the scheduler). For instance, the basic modelling of a periodic task requires at least two stopwatches for both models: one for the periodic activation, one for the progress of the task itself. However, in the product of SWA, all stopwatches are always used to define the state in the system whereas with SETPN, only the valuation of enabled transitions need to be considered. That means, for example, that when a periodic task is waiting for its periodical activation, only the stopwatch of the activator is required.

As a consequence, we create stopwatches on demand, *i.e.* when transitions are newly enabled. And when we create a new stopwatch, we reuse stopwatches that are no longer used, by always choosing the first stopwatch name available for new stopwatches. Furthermore, we use only one stopwatch for transitions for which valuations are equal for sure, *i.e.*, transitions that are enabled simultaneously, as long as they are running (or stopped) together. This is a situation that occurs fairly often.

Applying this policy for the creation of stopwatches allows us to obtain a state class stopwatch automaton with a fairly low number of stopwatches in practical cases.

3.6. DBM over-approximation

The over-approximation of domains by DBM can be used to compute extended state classes. This may lead to additional locations in the state class stopwatch automaton. However, as the guards and invariants are computed statically from the parameters of the SETPN itself, these additional locations are not reachable. As a consequence, the relation $\mathcal{R}$ of theorem 1 is also a bisimulation between the over-approximated state class stopwatch automaton and the SETPN:

Theorem 2 (Bisimulation) *Let $Q_{\mathcal{T}}$ be the set of states of the SETPN $\mathcal{T}$ and $Q_{\mathcal{A}}$ the set of states of the DBM over-approximated state class stopwatch 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 $\forall s = (M_{\mathcal{T}}, \nu_{\mathcal{T}}) \in Q_{\mathcal{T}}, \forall a = (l, \nu_{\mathcal{A}}) \in Q_{\mathcal{A}}, s\mathcal{R}a \Leftrightarrow M_{\mathcal{T}} = M_{\mathcal{A}}$ if $M_{\mathcal{A}}$ is the marking of the extended state class l and $\forall t \in enabled(M_{\mathcal{T}})$, $\exists x_t \in X, \nu_{\mathcal{T}}(t) = \nu_{\mathcal{A}}(x_t)$ and $\dot{x} = 1$ if t is active and $\dot{x} = 0$ otherwise.*

$\mathcal{R}$ is a bisimulation.

The proof is the same as for the exact computation. Indeed, as a convincer, 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 while 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 SETPN 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 SETPN, $\beta(t') \geq \nu_{\mathcal{T}}(t')$, this gives $\alpha(t) - \nu_{\mathcal{A}}(x_t) > 0$. On the other hand, by definition of the guards of the state class SWA, δ is true is equivalent to $\alpha(t) - \nu_{\mathcal{A}}(x_t) \leq 0$. With the preceding statement we can conclude that the guard δ is false, so l' is not reachable.

As a conclusion we can compute the state class stopwatch 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. This makes the cost of the translation low compared to that of the verification. But we can benefit from the expressivity and ease of use of the SETPN model and the state class SWA is, in general, easier to verify than a direct model using a product of SWA, because it has less stopwatches.

4. Case study

Following the ISO 11783 "Agriculture and Forestry" standard, which is based on SJAE J1939 (CAN Format Version 2.0B), some agricultural vehicle makers begin to use the CAN bus v2.0B and the Agricultural Bus System (LBS). Among them is FENDT, whose Electronic Control Unit (ECU) for the Vario 400 tractor is based on the INFINEON C167 processor.

In this section, we present experimental results based on a partial academic model for the control of the oscillation compensator (hydraulic shock absorber) and for the control of the differential blocking on a tractor with a sowing trailer.

Our simplified system consists of three processors running a real-time operating system (RTOS) and linked together with a CAN bus. A more complete description and the corresponding modelling can be found in [13]. In this case study, all tasks are periodic but it would be very easy to add aperiodic or sporadic tasks.

We have implemented a prototype for the translation of a SETPN into a SWA in the tool ROMEO (`http://www.irccyn.ec-nantes.fr/irccyn/d`

3 Stopwatches in a SETPN are actually the valuations of transitions

	Description		Direct SWA Modelling			Our method (SETPN $\xrightarrow{\text{ROMEO}}$ SWA $\xrightarrow{\text{HYTECH}}$ state-space)				
Ex.	Proc.	Tasks	SWA's	Sw.	HYTECH Time	Loc.	Trans.	Sw.	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

`/en/equipes/TempsReel/logs/software-2-romeo`), which gives the resulting SWA in the HYTECH input format. The state space of this SWA is then computed with HYTECH (forward computation).

We compared the efficiency of our method with a generic direct modelling with HYTECH on this case study. We also tested several simpler and more complex related systems obtained by removing or adding tasks and/or processors. Table 1 gives the obtained results.

Columns 2 and 3 give the number of processors and tasks of the system. Columns 4, 5 and 6 describe the direct modelling in HYTECH results by the number of SWA of the product, the number of stopwatches and the time taken by HYTECH to compute the state space. For this generic modelling, we basically used the product of one SWA per task and one SWA for each scheduler. We also used an "optimization" that consists of sharing some of the periodic activation clocks whenever possible. Columns 7, 8, 9 give the results for our method. We give the number of locations, transitions and stopwatches of the SWA 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 SWA generated by our method. Times are given is seconds and NA means that the HYTECH computation could not yield a result on the machine used.

These computations have been made on a POWERPC G4 1.25GHz with 500Mo of RAM.

We see that the computation on a direct modelling as a product of SWA is quickly intractable (Example 3). However, with our method, we are able to deal with systems of much greater size. With the last example, the computation is still possible with ROMEO but the state space of the resulting SWA is not computable anymore. In this case, for safety properties, we can exploit directly the DBM-based extended state class graph generated by ROMEO by using classical methods like observers for instance, but keeping in mind that this is an over-approximation.

For this case-study, we computed the whole state-space of the model but we can also check specific timed properties, including schedulability. We can compute the worst case response time of a task, for instance, by adding an observer which resets a clock on the firing of the first and last transitions of the SETPN model of that task.

5. Conclusion

In this paper, we have given a method for computing the state space of a scheduling extended time Petri net as a stopwatch automaton. This is beneficial in several areas: modelling real-time concurrent systems with SETPN is very natural, the state class stopwatch automaton can be verified using a well-known tool on hybrid linear automata: HYTECH. This method leads to a single stopwatch automaton with fewer stopwatches than in the product of stopwatch automata obtained through a generic direct modelling with SWA. So, the verification of properties using HYTECH is more likely to be tractable. This approach is coherent and efficient because the translation can be done by using a fast DBM based over-approximating algorithm, while still having a result SWA that is timed-bisimilar to the SETPN. So the cost of the translation is fairly lower than the verification of properties on a direct modelling as a product of SWA. Practical experimentations show that our method greatly increases the size and complexity of the systems for which the state space can be computed with HYTECH.

Further work includes the extension of the SETPN model for the round-robin and dynamic scheduling policies such as Earliest Deadline First, and adaptation of the method to the inhibitor hyperarcs time Petri nets [22] model.

References

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

[2] B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using time Petri nets. *IEEE transactions on software engineering*, 17(3):259–273, 1991.

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

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

[5] F. Cassez and K. Larsen. The impressive power of stopwatches. In C. Palamidesi, editor, *11th International Conference on Concurrency Theory, (CONCUR'2000)*, number 1877 in Lecture Notes in Computer Science, pages 138–152, University Park, P.A., USA, July 2000. Springer-Verlag.

[6] G. B. Dantzig. Linear programming and extensions. *IEICE Transactions on Information and Systems*, 1963.

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

[8] E. Fersman, P. Petterson, and W. Yi. 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)*, volume 2280 of *Lecture Notes in Computer Science*, pages 67–82, Grenoble, France, 2002. Springer-Verlag.

[9] M. Harbour, M. Klein, and J. Lehoczky. Fixed priority scheduling of periodic tasks with varying execution priority. In *12th IEEE Real-Time Systems Symposium (RTSS'91)*, pages 116–128, San Antonio, USA, december 1991. IEEE Computer Society Press.

[10] T. Henzinger, P.-H. Ho, and H. Wong-Toi. Hytech: A model checker for hybrid systems. *Journal of Software Tools for Technology Transfer*, 1(1-2):110–122, 1997.

[11] P.-E. Hladik and A.-M. Déplanche. Analyse d'ordonnançabilité de tâches temps-réel avec offset et gigue. In *11th international Conference on Real-Time Systems (RTS'03)*, pages 307–332, Paris, France, april 2003.

[12] K. G. Larsen, P. Pettersson, and W. Yi. Model-checking for real-time systems. In *Fundamentals of Computation Theory*, pages 62–88, 1995.

[13] D. Lime and O. Roux. State class stopwatch automaton of a scheduling extended time Petri net for the verification of real-time systems. Technical report, Institut de Recherche en Communications et Cybernétique de Nantes (IRCCyN), 2004.

[14] D. Lime and O. H. Roux. Expressiveness and analysis of scheduling extended time Petri nets. In *5th IFAC International Conference on Fieldbus Systems and their Applications, (FET'03)*. Elsevier Science, July 2003.

[15] D. Lime and O. H. Roux. State class timed automaton of a time Petri net. In *10th International Workshop on Petri Nets and Performance Models, (PNPM'03)*. IEEE Computer Society, Sept. 2003.

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

[17] P. M. Merlin. *A study of the recoverability of computing systems*. PhD thesis, Department of Information and Computer Science, University of California, Irvine, CA, 1974.

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

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

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

[21] O. H. Roux and A.-M. Déplanche. A t-time Petri net extension for real time-task scheduling modeling. *European Journal of Automation (JESA)*, 36(7), 2002.

[22] O. H. Roux and D. Lime. 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'04)*, Bologna, Italy, june 2004. Lecture Notes in Computer Science.

[23] K. Tindell. *Fixed priority scheduling of hard real-time systems*. PhD thesis, Department of Computer Science, University of New York, 1994.