A Framework for Scheduler Synthesis

K. Altisen, G. Gößler, A. Pnueli, J. Sifakis, S. Tripakis, and S. Yovine
Verimag, Centre Équation, 2 Ave. de Vignate, 38610 Gières, France
{altisen,goessler,pnueli,sifakis,tripakis,yovine}@imag.fr

Abstract

In this paper we present a framework to integrate specification and scheduler generation for real-time systems. In a first step, the system, which can include arbitrarily designed tasks (cyclic or sporadic, with or without precedence constraints, any number of resources and CPUs, ...) is specified in terms of a timed Petri-net formalism. We show how to generate, in a second step, the most general non-preemptive dynamic online scheduler for this specification, using a controller synthesis technique.

1 Introduction

A complex real-time system is typically composed of several tasks that interact. The execution of each task is subject to different kinds of constraints, some of which are proper to the task such as completion times, deadlines and periods, while others, such as resource sharing and synchronization delays, are imposed by the environment in which the tasks are executed.

An important problem that arises is the one of ensuring that the tasks can be executed in such a way that (1) they respect the constraints they are subject to, and (2) the overall system performs correctly with respect to its specification, i.e., satisfies a given property. Solving this problem usually requires scheduling the tasks in an appropriate way. This amounts to, given a system S and a property P, constructing a scheduler C (that depends on S and P) which "controls" the execution of S so it satisfies P.

One approach to constructing a scheduler C consists in giving a scheduling policy, that is, a systematic way of ordering the execution of the tasks in the time line according to their deadlines, periods and priorities, for a specific property P, e.g., mutual exclusion [12, 4]. This approach, which has been thoroughly studied in the real-time systems' literature, provides means to schedule systems that satisfy sufficient schedulability conditions depending on a particular scheduling policy.

In this paper, we follow a different approach which consists in building a scheduler (if such a scheduler exists) tailored to the particular application and property, regardless of any a-priori fixed scheduling policy which might not fit the particular requirements and structure of both the system and the specification. In other words, the scheduler needs not be constrained to follow a predefined scheduling policy but may adapt its decisions according to the behavior of the environment and the property to be satisfied. In this sense, constructing a scheduler amounts to doing controller synthesis [16].

The approach based on scheduling policies is well suited for real-time applications (e.g., avionics) where the behavior of the environment is not predictable and reactions to external events must be immediate. There are, however, many reasons in favor of following our approach, especially for real-time applications, such as multimedia and telecommunication systems, that interact with strongly constrained environments. For such applications, it is desirable to generate tailor-made schedulers at compile time that make optimal use of the underlying execution hardware and shared resources, guided by the knowledge of all the possible behaviors of the environment.

To achieve our goal, we propose a framework for constructing schedulers that combines recent results on modeling and synthesis of timed systems. This framework is supported by a prototype tool whose structure is depicted in fig. 1.

As modeling language we use Petri Nets (PND) equipped with a set of clocks that measure time as in the timed automata formalism [1]. For methodological reasons, both to facilitate the description of synchronization conditions and to enhance the readability of the specifications, we enrich the basic formalism with high-level synchronization primitives. Besides, the transitions of a PND are classified into *controllable* and *uncontrollable*. Intuitively, the former correspond to transitions that can be triggered by the scheduler (e.g., grant access to a shared resource), while the latter are subject to timing constraints imposed by the environment (e.g., completion of a task, communication delay). The semantics of a PND is given in terms of an appropriate class of timed automata, called timed automata with deadlines (TAD).

The backbone of our approach is an algorithm that given a TAD $\mathcal{A}$ and a property P, constructs a TAD $\mathcal{A}_P$ which models all the behaviors of $\mathcal{A}$ that satisfy P for any possible sequence of uncontrollable transitions. In other words, $\mathcal{A}_P$ describes all the schedules that satisfy the property, a schedule being a sequence of controllable transitions for a given pattern of uncontrollable behaviors.

The paper is organized as follows. Section 2 presents the description formalism (PND) and the semantic model (TAD). Section 3 is devoted to the synthesis algorithm. To demonstrate the feasibility of the approach we treat several case-studies issued from different application domains (Section 4). In the last section we present some conclusions and discuss future and related work.

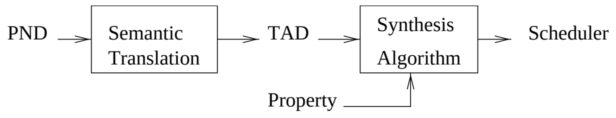


Figure 1. The framework.

2 Modeling

2.1 Description formalism

As formalism to describe the behavior of real-time systems we have chosen *Petri Nets with Deadlines* (PND) [5]. A PND consists of (1) a 1-safe Petri net $(\mathcal{P},\mathcal{T},A)$ where $\mathcal{P}$ is a finite set of places, A is a finite vocabulary of actions, and $\mathcal{T} \subseteq 2^{\mathcal{P}} \times A \times 2^{\mathcal{P}}$ is a transition relation; (2) a set $X = \{x_1,\ldots,x_m\}$ of real-valued variables called clocks ranging in $\mathbb{R}_+$; (3) a labeling function h mapping untimed transitions $(P,a,P') \in \mathcal{T}$ into timed transitions: $h(P,a,P') = (P,(a,g,r),P')$, where the guard g is a predicate defined by the following grammar: $g ::= x\#c \mid x-y\#c \mid g \wedge g \mid \neg g$, where $x,y \in X$, c is an integer and $\# \in \{\leq,<\}$, and $r \subseteq X$ is a set of clocks to be reset.

We assume that $A = A^c \cup A^u$ is partitioned into two classes of actions. Transitions labeled with actions in A^c are called *controllable* while those labeled with actions in A^u are called *uncontrollable*. The former are those that can be triggered by the scheduler (e.g., grant a resource to a process), whereas the latter only depend on the behavior of the environment (e.g., jitter, communication delays).

A transition in a PND is enabled for a given marking and clock valuation (i.e., an assignment of real values to clocks) iff it is enabled for the marking (according to the semantics in the untimed case), and the clock valuation satisfies the associated guard.

Example 2.1 Consider a system composed of two jobs Job_1 and Job_2 [11]. Let e_i and s_i, $i = 1,2$ be Job_i's execution time and start time, respectively. The behavior of the system is constrained to the following conditions (figure 2): (1) $5 \leq e_1 \leq 7$ and $3 \leq e_2 \leq 4$; (2) $s_1+e_1 \leq 12$ and $s_2+e_2 \leq 25$; (3) $s_2+e_2 \leq s_1+e_1+10$; and (4) $s_2 \geq 14$. The termination of the execution of a job is an uncontrollable action, while the start time of a job can be controlled to meet the requirements.

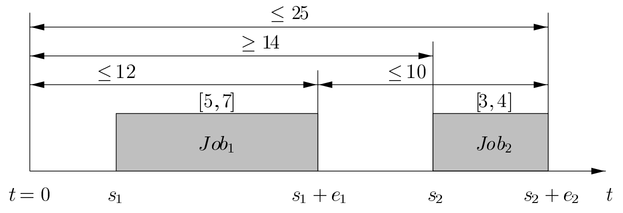


Figure 2. Example of two jobs.

We model the system as the PND shown in figure 3. Guards of controllable (resp. uncontrollable) transitions are marked c (resp. u). The set of clocks to be reset at a transition is given between accolades, if not empty. Time is measured by the clock t. s_1 (resp. s_2) is the value of t when Job_1 (resp. Job_2) starts, that is, when the controllable transition beg_1 (resp. beg_2) is fired. s_1+e_1 (resp. s_2+e_2) is the value of t when Job_1 (resp. Job_2) finishes, that is, when the uncontrollable transition end_1 (resp. end_2) is fired. a measures the time elapsed since the end of Job_1. If any of the above conditions is not respected, the system reaches the *error* state by some of the err_i transitions for $i = 1,2,3$.

2.2 Semantic model

As semantic model we have chosen the so-called *Timed Automata with Deadlines* (TAD) [5]. TAD's are timed automata where invariant conditions associated to control locations, typically used to impose upper bounds on the values of the clocks and therefore force the automaton to take a transition, have been replaced by deadline conditions on the transitions which must be taken before the deadlines expire.

A TAD consists of (1) a discrete labeled transition system $(S,\rightarrow,A)$ with S a finite set of discrete states, A

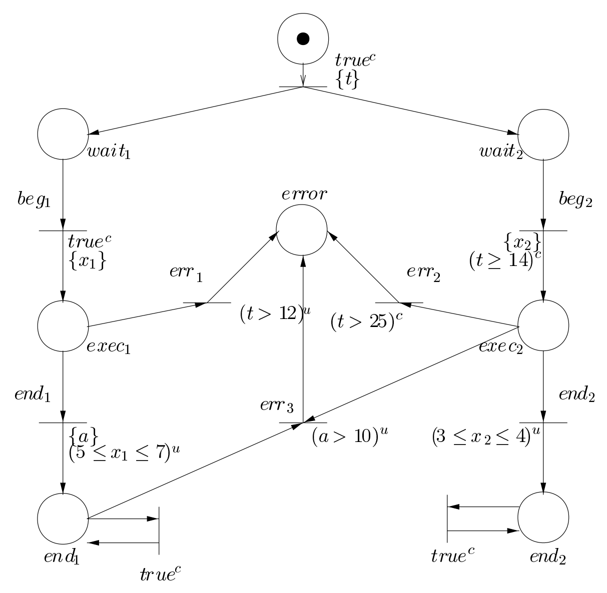


Figure 3. PND of the two jobs system.

a finite vocabulary of actions, and $\rightarrow \subseteq S \times A \times S$ an untimed transition relation; (2) a set $X = \{x_1, \ldots, x_m\}$ of real-valued variables called *clocks* ranging in $\mathbb{R}_+$; and (3) a labeling function h mapping *untimed transitions*, elements of $\rightarrow$, into *timed transitions*: $h(s, a, s') = (s, (a, g, r), s')$, where g is the guard and $r \subseteq X$ is a set of clocks to be reset.

Formally, the semantics of a PND is defined in terms of the semantics of a TAD. A PND $(\mathcal{P}, \mathcal{T}, A, X, h)$ defines a TAD $(S, \rightarrow, A, X, h')$ such that (1) $S \subseteq 2^{\mathcal{P}}$ is a set of discrete states (markings); (2) $s \xrightarrow{a} s'$ iff there exist $P_1, P_2 \subseteq \mathcal{P}$ such that $(P_1, a, P_2) \in \mathcal{T}$, $P_1 \subseteq s$ and $s' = (s - P_1) \cup P_2$; and (3) $h'(s, a, s') = h(P_1, a, P_2)$.

Now, a state of a TAD is a pair (s, v), where $s \in S$ is a discrete state and $v \in \mathbb{R}_+^m$ is a clock valuation. We associate with a TAD a transition relation $\rightarrow \subseteq (S \times \mathbb{R}_+^m) \times (A \cup \mathbb{R}_+) \times (S \times \mathbb{R}_+^m)$. Transitions labeled by elements of A correspond to *discrete state changes* while transitions labeled by non-negative reals correspond to *time steps*.

Given $s \in S$, let J be the set of indices such that $\{(s, a_j, s_j)\}_{j \in J}$ is the set of all the transitions departing from s. Also let $h(s, a_j, s_j) = (s, (a_j, g_j, r_j), s_j)$. For all $j \in J$, $(s, v) \xrightarrow{a_j} (s_j, v[r_j])$ iff $g_j(v)$ evaluates to true and $v[r_j]$ is the variable valuation obtained from v when all the clocks in r_j are set at zero and the others left unchanged. For all $t \in \mathbb{R}_+$, $(s, v) \xrightarrow{t} (s, v + t)$, where $v + t$ is the valuation obtained from v by increasing all the clock values by t, iff the following condition holds:

$$\begin{array}{l} \left(\forall t', 0 \leq t' < t \,.\, \neg \bigvee_{j \in J} d_j^c(v + t')\right) \quad \wedge \\ \left(\forall t', 0 < t' \leq t \,.\, \neg \bigvee_{j \in J} d_j^o(v + t')\right) \end{array} \qquad (1)$$

where d_j^c and d_j^o are, respectively, the *closed* and *open deadlines* of transitions j defined by

$$\begin{array}{rcl} d_j^c(v) & = & g_j(v) \wedge \exists t > 0. \forall t' \in (0; t]. \neg g_j(v + t') \\ d_j^o(v) & = & \neg g_j(v) \wedge \exists t > 0. \forall t' \in (0; t]. g_j(v - t') \end{array}$$

The closed deadline d_j^c is contained in the guard g_j and can be reached, but not left behind by time progress. On the other hand, the open deadline d_j^o is not contained in the guard g_j and cannot be reached by time progress although it can be approached arbitrarily close. Thus, condition (1) establishes that it is not allowed that the progress of time disables any enabled transition departing from s.

Notice that the definition of time progress makes sure that the TAD is non-blocking, that is, every state (s, v) has *at least* one outgoing transition, either discrete or timed. This is, indeed, the main semantic property that distinguishes TAD's from TA's: in contrast to invariant conditions associated with discrete states that impose upper bounds on clocks and force the TA to take a transition, deadline conditions associated with transitions permit the modeling of *urgency* while avoiding introducing deadlocks as a side effect [5].

2.3 Synchronization modes

A synchronization transition (one with more than one input arc) can be considered as the coordination of transitions corresponding to its input arcs. Synchronization guards can be obtained from guards of those transitions by applying synchronization rules characterizing the semantics of the synchronization. We adopt the following notation: for synchronization transitions, the guards are replaced by guards g_i associated with their input arcs, and a label denoting a synchronization mode. The latter corresponds to a type of coordination and defines a way for composing the guards of the input arcs to obtain the synchronization guard (figure 4), where *mode* is one of *AND*, *MAX*, *MIN*.

For the description of the synchronization modes we define $\Diamond g$ (read "eventually g") and $\diamondminus g$ (read "once g") as follows:

$$\begin{array}{lll} (\Diamond g)(v) & \text{if} & \exists t \in \mathbb{R}_+ .\ g(v + t) \\ (\diamondminus g)(v) & \text{if} & \exists t \in \mathbb{R}_+ .\ \exists v' \in \mathbb{R}_+^m . \\ & & v = v' + t \wedge g(v') \end{array}$$

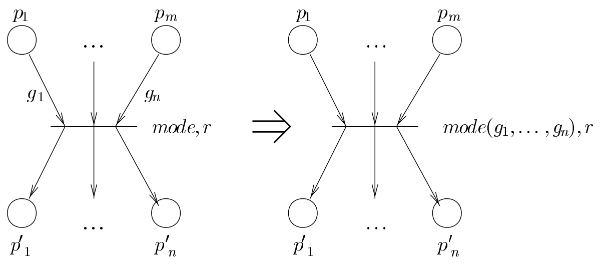


Figure 4. Synchronization notation.

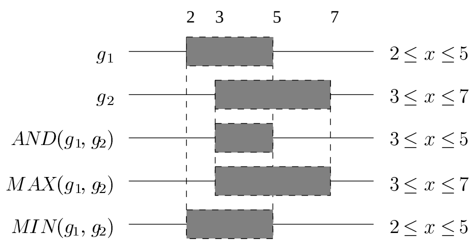


Figure 5. Resulting guards for the three synchronization modes.

Notice that the operators $\Diamond$ and $\diamondminus$ can be eliminated to obtain simple predicates without quantifiers. We use these operators to define the following synchronization modes:

AND-synchronization. The resulting guard g is the conjunction $g = \bigwedge_{i \in [1 \dots n]} g_i$ of the input guards. This simply means that synchronization is possible only if all processes can terminate together. In the example of figure 5, we get $g = g_1 \wedge g_2 = 3 \le x \le 7$.

MAX-synchronization (rendez-vous). Synchronization can take place only if all the contributing processes have terminated. This implies synchronization at times t bounded by the maximum of the earliest termination times and the maximum of the latest termination times of the contributing processes. For this synchronization mode, we take $g = \bigvee_{i \in [1 \dots n]} g_i \wedge \bigwedge_{j \neq i} \diamondminus g_j$. The i-th term of the guard means that the i-th process can terminate now while the others can either terminate now or have already terminated. In the example of figure 5, we get $g = g_1 \wedge (\diamondminus g_2) \vee (\diamondminus g_1) \wedge g_2 = 3 \le x \le 7$.

MIN-synchronization. Synchronization takes place when one of the contributing processes terminates and the others will eventually terminate. This corresponds to a kind of interrupt where the fastest process triggers the synchronization transition even though the other processes have not terminated. Notice that synchronization times t are bounded by the minimum of the earliest and the minimum of the latest termination time of the contributing processes. We take $g = \bigvee_{i \in [1 \dots n]} g_i \wedge \bigwedge_{j \neq i} \Diamond g_j$. The i-th term of the guard means that the i-th process can terminate now and all the others can terminate either now or in the future. In the example of figure 5, we get $g = g_1 \wedge \Diamond g_2 \vee \Diamond g_1 \wedge g_2 = 2 \le x \le 5$.

When only controllable actions are composed, synchronization is supposed to model coordination, therefore, the resulting action is also controllable. That is, $mode(g_1^c, g_2^c) = (mode(g_1, g_2))^c$. This can be easily understood by the fact that the dates can be appropriately chosen for each action so as to achieve synchronization. When controllable and uncontrollable actions are composed, it is not in general possible to associate one of the types, controllable or uncontrollable, to the synchronization guard. However, synchronization involving uncontrollable actions can be modeled using more than one transition [7]. As an example, the MIN-synchronization between a controllable and an uncontrollable action can be modeled by putting the corresponding transitions in parallel, such that they share the same input and output places (see the following example). Discussing other synchronization primitives involving uncontrollable actions is out of the scope of this paper.

All the defined modes are commutative and associative, which allows us to extend them in order to synchronize n actions. AND-synchronization is the most usual synchronization mode considered in literature. However, the use of MAX and MIN allows more concise specifications and avoids explosion of the state space [5]. Furthermore, it allows extending untimed specifications to timed specifications without modifying the underlying control structure. To illustrate the use of the defined synchronization modes consider the following example.

Example 2.2 A multimedia document is composed of the following basic activities and their corresponding duration constraints, noted as [minimal duration, maximal duration]: music [30, 40], video [15, 20], audio [20, 30], text [5, 10], applet [20, 30], picture [20, ∞]. In the beginning, music, video, audio, and applet are launched in parallel. The basic activities are submitted to the following synchronization constraints: (1) video and audio terminate as soon as any one of them ends; their termination is immediately followed by the text to be displayed; (2) music and text must terminate at the same time; (3) the applet is followed by a picture; (4) the document terminates as soon as both the picture and the music (and text) have terminated; and (5) the execution times of both the audio and the applet depend on the machine load and are therefore uncontrollable. Fig. 6 shows a PND modeling the described document.

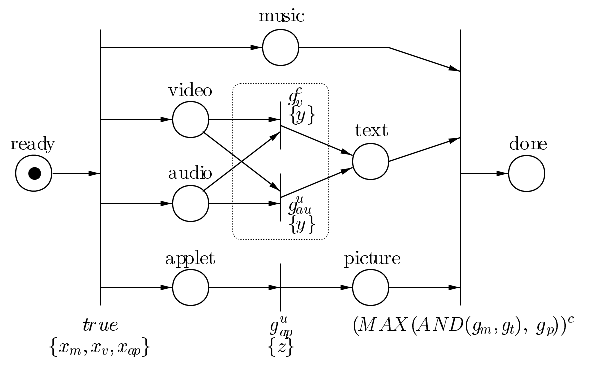


Figure 6. PND of the multimedia document.

Transitions synchronizing controllable and uncontrollable actions are highlighted by a dotted box. The guards are as follows: $g_m = (30 \leq x \leq 40)$, $g_v = (15 \leq y \leq 20)$, $g_{au} = (20 \leq y \leq 30)$, $g_t = (5 \leq y \leq 10)$, $g_{ap} = 20 \leq z \leq 30$, $g_p = 20 \leq z$, and

$$\begin{aligned} g^c &:= MAX(AND(g_m^c, g_t^c),\ g_p^c) \\ &= (g_m \wedge g_t \wedge \diamondsuit g_p \ \vee\ \diamondsuit (g_m \wedge g_t) \wedge g_p)^c \\ &= (30 \leq x \ \wedge\ 5 \leq y \ \wedge\ 20 \leq z \ \wedge \\ &\quad 20 \leq x - y \leq 35 \ \wedge\ x - z \leq 40 \ \wedge \\ &\quad y - z \leq 10)^c \end{aligned}$$

The corresponding TAD is depicted in figure 7.

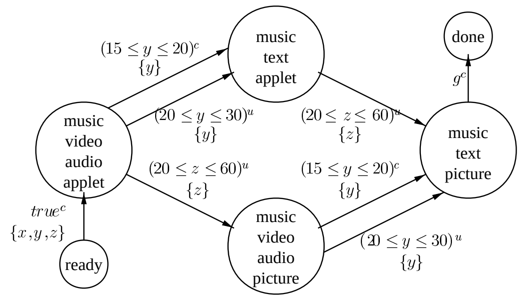


Figure 7. TAD of the multimedia document.

A tool to synchronize transitions of a PND, simplify the resulting guards, and translate the obtained PND into a TAD, has been implemented [7]. All the examples treated in this paper have been processed with this tool.

3 Synthesis

The synthesis algorithm allows to compute from a given TAD $\mathcal{A}$ and a given property Q a TAD A_Q characterizing all the states of $\mathcal{A}$ that satisfy Q regardless of the uncontrollability of some transitions. In other words, A_Q characterizes all the sequences of controllable transitions that keep the system only in states where Q is satisfied for any uncontrollable behavior.

The synthesis algorithm is an adaptation of the one proposed in [14] for properties of the form $\Box P$ (read "always P") and $\Diamond P$ (read "eventually P") where P is a state predicate. They are based on a procedure that starts with the states that satisfy P and keeps on iterating a single-step controllable predecessor operator π until a fixed point is reached. The iteration schemes are shown in table 1. It has been shown that the iterations terminate [3].

(a)	(b)
$Q^0 = P$	$Q^0 = P$
Repeat	Repeat
$\quad Q^{i+1} = Q^i \cup \pi(Q^i)$	$\quad Q^{i+1} = Q^i \cap \pi(Q^i)$
Until $Q^i = Q^{i+1}$	Until $Q^i = Q^{i+1}$

Table 1. Synthesis algorithms for $\Diamond P$ (a) and $\Box P$ (b)

The operator π is defined as follows. Given a TAD, $(S, \rightarrow, A, h)$ and a state predicate P, we define the predicate transformer π such that $\pi(P)$ represents all the states of the TAD from which it is possible to reach a state of P by taking some controllable transition, possibly after letting time pass, while ensuring that there is no uncontrollable transition that leads into $\neg P$. More formally, for a state (s, v) of the TAD:

$$\begin{aligned} \pi(P)(s, v) &= \exists t \in I\!R_+ .\ (s, v) \xrightarrow{t} (s, v + t) \wedge \\ &\quad pre_c(P)(s, v + t) \wedge \\ &\quad \neg \exists t' \in [0, t].pre_u(\neg P)(s, v + t') \end{aligned}$$

where $pre_c(P)$ is the set of states from which a state of P can be reached by executing a controllable transition:

$$\begin{aligned} pre_c(P)(s, v) &= \exists s \xrightarrow{a} s'.\ h(s, a, s') \\ &= (s, (a, g, r), s') \wedge P(s', v[r]) \wedge \\ &\quad (a \in A^c \wedge g(v) \\ &\quad \vee a \in A^u \wedge d_g^c(v)) \end{aligned}$$

where we consider as controllable the transitions produced when an uncontrollable transition reaches a closed deadline[1]. Notice that the definition of $pre_c(P)$

[1] In a similar way, open deadlines of uncontrollable actions can be considered as controllable, but the current implementation does not treat this case.

characterizes only those states that have a successor reachable by a controllable transition, and henceforth ensures the non-blocking property. $pre_u(P)$ is the set of states from which a state of P can be reached by executing an uncontrollable transition:

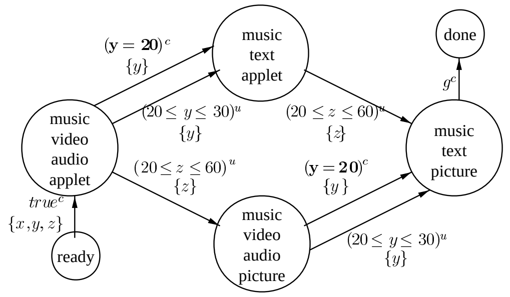
$$\begin{aligned} pre_u(P)(s,v) &= \exists s \xrightarrow{a} s'.\ h(s,a,s') \\ &= (s,(a,g,r),s') \wedge P(s',v[r]) \wedge \\ &\ a \in A^u \wedge g(v). \end{aligned}$$

Now, let Q be the property and $Q^* = \bigcup_{s \in S} Q^*_s$ be the set of states computed by one of the algorithms above. $\mathcal{A}_Q$ has the same discrete structure as $\mathcal{A}$, and the same timing information except for its controllable guards: the guard g of every controllable transition $(s,(a,g,r),s')$ is replaced by $g' = g \wedge Q^*_s \wedge pre_a(Q^*_{s'})$, where $pre_a(P)(s,v) = P(s',v[r]) \wedge g(v)$, while uncontrollable transitions remain unchanged. Clearly, if $\mathcal{A}_Q$ is initialized with a state in Q^*, by executing controllable transitions, it will remain in Q^* and cannot possibly reach states of $\neg Q^*$ by executing uncontrollable actions. $\mathcal{A}_Q$ is *maximal* in the sense that it spans all the possible schedules for the considered properties [3]. Notice that if Q^* is empty, then no scheduler could prevent $\mathcal{A}$ from violating Q.

The algorithms for computing Q^* and $\mathcal{A}_Q$ for reachability and invariance properties have been implemented in KRONOS [6] where special care has been taken to reduce complexity in the implementation of π [2]. The results and performances of the algorithms in several case studies are given in Section 4.

Example 3.1 Recall the example about the two jobs presented in Section 2. The goal is to synthesize a scheduler that starts Job_1 and Job_2 at appropriate times to avoid reaching *error* state. We apply the synthesis algorithm for the property $\Box\neg error$. The result establishes that the only possible schedules must execute the two processes in mutual exclusion, that is, Job_1 must be started after Job_2 has finished as suggested by figure 2. If we consider the places $wait_1$ and $wait_2$, we find out that the schedulable states satisfy the following constraints: $3 \le s_1 \le 5$ and $14 \le s_2 \le 18$ and $s_2 - s_1 \le 13$. In words, the start time of Job_1 must be chosen in the interval $[3,5]$ and Job_2 must be started no longer than 13 time units after Job_1. The constraint synthesized by our algorithm contains all the solutions exhibited in [11].

Example 3.2 Here we illustrate the application of the synthesis algorithm for reachability properties on the multimedia example. In this case, we seek for the existence of a scheduler that steers the system from the initial state to the state *done*. We apply the algorithm for the property $\Diamond done$ and the result obtain is that the document is indeed schedulable. The execution time of *text* can be dynamically adapted to the duration of *video* and *audio* so as to make *music* and *text* terminate synchronously. The corresponding scheduler is shown in fig. 8. The restricted guards of controllable transitions, computed by the synthesis algorithm, are printed in bold. Notice that if *video* terminates at a time $y < 20$, the marking {music, text, applet} will be reached with a valuation satisfying $x - y < 20$ which falsifies the synchronization guard g^c and therefore the only possible schedule guaranteeing the reachability of *done* must terminate *video* at $y = 20$.



music text applet
done
music video audio applet
music text picture
music video audio picture
ready
$(\mathbf{y = 20})^c$ $\{y\}$
$(20 \le y \le 30)^u$ $\{y\}$
$(20 \le z \le 60)^u$ $\{z\}$
g^c
$(20 \le z \le 60)^u$ $\{z\}$
$(\mathbf{y = 20})^c$ $\{y\}$
$true^c$ $\{x,y,z\}$
$(20 \le y \le 30)^u$ $\{y\}$

Figure 8. Multimedia scheduler.

4 Case studies

In this section we apply the proposed synthesis framework to three case studies.

4.1 The Greeting Card Example

We consider here an interactive greeting-card consisting of basic media such as texts, audio tracks, a picture and an animation. The original specification has been provided in MADEUS [9]. We model the card as a PND and then synthesize a controller that generates execution scenarios respecting the temporal constraints of the specification.

The basic activities, along with their possible execution times and the controllability of their termination, are the following ones (C (resp. U) means that the document is controllable (resp. uncontrollable)):

start	button	$[0,100]$	U
xmast	text “Merry Christmas”	$[4,10)$	C
xmasm	audio “Jingle Bells”	$[5,7]$	U
and	text “and”	$[2,5]$	C
smiley	animation	$[3,4]$	U
family	text “family”	$[5,15]$	C
yeart	text “Happy new year”	$[5,15]$	C
smith	text “Smith”	$[5,15]$	C
yearm	audio “Firework music”	$[5,7]$	C
photo	picture	$[20,\infty)$	C

With a click on the *start* button, the text “Merry Christmas” appears, accompanied by Christmas music. As soon as one of them has terminated playing, the texts “and” and “Happy New Year” are displayed one after the other. The latter is played together with Händel’s “Firework music” sound track. The first one to terminate interrupts the other, and the text “Smith” moves over the screen. During the displaying of the word “and”, an animated smiley is displayed. When it ends, the text “family” appears and moves over the screen. Finally, the two textual elements “family” and “Smith” come to stop synchronously, one above the other. While playing the described media, a family photo is displayed in the background.

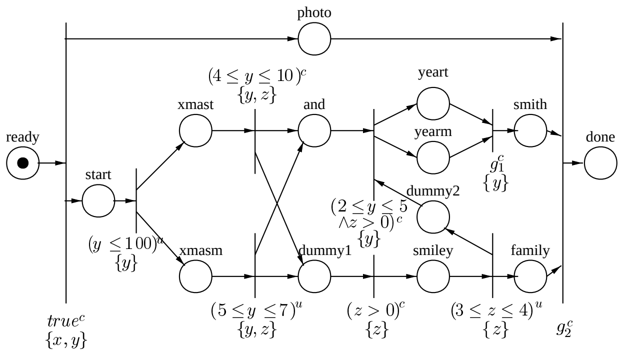


Figure 9. PND modeling the greeting card.

The corresponding PND is shown in figure 9. The places *dummy1* and *dummy2* make sure that *smiley* is executed during the displaying of AND. The MIN-synchronization between the outgoing transitions of *xmast* and *xmasm*, the latter of which is uncontrollable, is done according to the MIN-synchronization rule. The synchronization guards g_1^c and g_2^c are obtained as follows:

$$\begin{aligned}
g_1^c &= MIN((5 \leq y \leq 15)^c, (5 \leq y \leq 7)^c) \\
&= (5 \leq y \leq 7)^c \\
g_2^c &= AND(\ (x \geq 20)^c, \\
&\qquad AND(\ (5 \leq y \leq 15)^c, \\
&\qquad\qquad (5 \leq z \leq 15)^c\ )\ ) \\
&= (x \geq 20 \wedge 5 \leq y \leq 15 \wedge 5 \leq z \leq 15)^c
\end{aligned}$$

where g_1^c synchronizes *yeart* and *yearm* according to the MIN synchronization rule, and g_2^c synchronizes *photo*, *smith* and *family*, which are to terminate simultaneously.

The obtained PND is translated into its corresponding TAD shown in fig. 10. We use the synthesis algorithm to construct a scheduler that guarantees that the state *done* is eventually reached from the state *ready*. Notice that since there is no choice on the possible

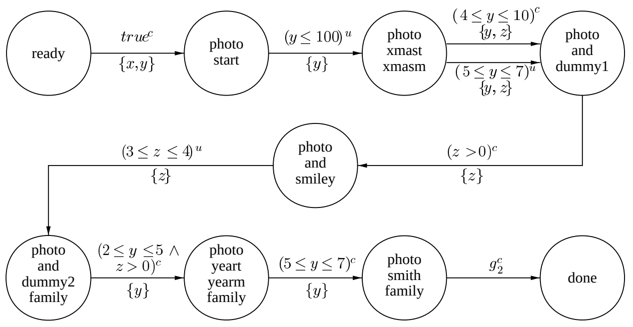


Figure 10. Greeting card specification translated into a TAD.

controllable transitions, scheduling consists in appropriately choosing the delays of the controllable basic media according to the delays chosen by the environment for the uncontrollable ones. The synthesis algorithm for $\Diamond$*done* outputs the TAD shown in fig. 11. The synthesized guards are written in bold.

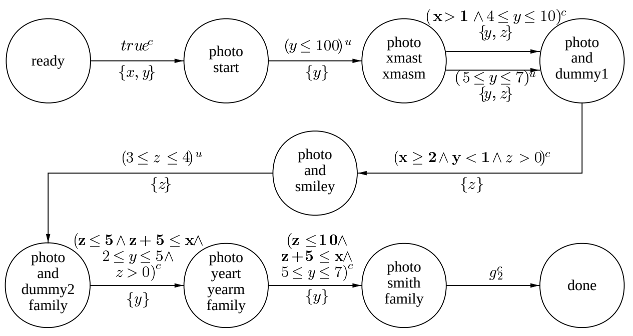


Figure 11. Controller obtained by synthesis.

4.2 A system of three tasks

We consider here a system of three processes P^1, P^2 and P^3 sharing three non-preemptable resources R_0, R_1 and R_2, as described in [13]. P^1 is a periodic process of period equal to 9 and deadline equal to 8. After an initial jitter of at most 1, P^1 uses R_0 for some time between 2 and 3 and later R_0 and R_1 for a time between 1 and 2. P^1 releases R_0 between the two utilisations. P^2 is an aperiodic process with a minimum inter-arrival time of 10. It simultaneously uses resources R_0 and R_2 during a time ranging in $[1,2]$. P^3 is an aperiodic process with a minimum inter-arrival time of 6. It needs both resources R_1 and R_2 simultaneously to execute in a time ranging in $[1,2]$ (figure 12).

We model this system as the PND shown in fig. 13. We use three error places, named err_i with $i = 1,2,3$,

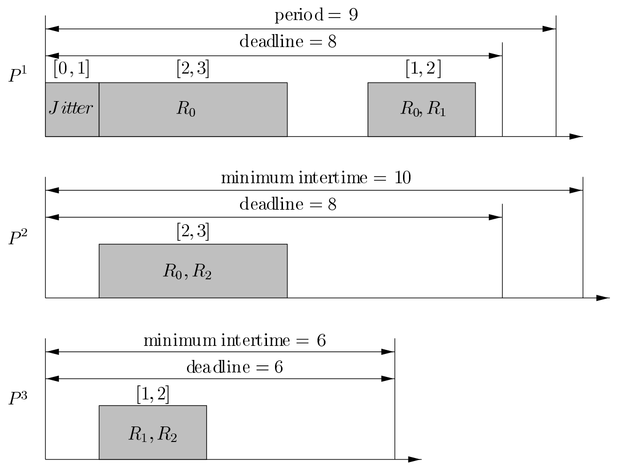


Figure 12. A system with three tasks.

which are reached when the respective deadlines cannot be met. This means that unreachability of err_i implies liveness of P^i. The size of the corresponding TAD is shown in table 2. The problem to be addressed consists in finding a scheduler that guarantees that every process meets its deadline (i.e., $\Box\neg(err_1 \vee err_2 \vee err_3)$).

In [13], P^1 and P^2 are assumed to be mandatory whereas P^3 is optional, i.e., can be rejected if the deadlines of the others are compromised. A scheduler is provided for the system composed of the two processes P^1 and P^2, but for the full system, no complete scheduler is given.

We have synthesized deadlock-free[2] schedulers for the two cases. The scheduler for the full system contains states from which the optional task P^3 is actually scheduled. Table 2 shows the size of the scheduler and the performances of the algorithm.

4.3 A robotic arm

This case study is also taken from [13]. It is about a robotic arm programmed to take objects from a conveyor belt, to store them in a buffer shelf, and to put them eventually into a basket. The arm is controlled by five tasks sharing one CPU:

- A *Trajectory Control* (TC) is spawned every 16ms to read commands from a shared buffer P_2 and issue set-points to the low-level arm controller. This task terminates immediately if there are no commands to process, otherwise it has a WCET between 5ms and 6ms. Its deadline is equal to the period of 16ms.

[2]The fact that each cycle in the PND lasts a time strictly greater than one guarantees non-zenoness.

- Two motion executers (ME), a *lifter* and a *putter*, are invoked whenever the system must move an item; each ME generates a set of command in the command buffer. The lifter is activated whenever the system detects an object on the conveyor belt, with a minimum intertime of 40ms between each arrival. The lifter issues to the TC the commands to get the object and to put it in the buffer shelf P_1, and activates the putter. The putter sends to the TC the commands to take the object from the buffer shelf and put it into the basket. Both ME produce their commands within 4ms to 8ms.
- A *Sensor Reader* (SR) reads several sensors every 24ms. Its execution time is 1ms, and its deadline is equal to the period. The results of the SR are used by the TC.
- A *Motion Planner* (MP) refines the motion plan for the MEs each time it can be run without compromising the safeness of the remaining tasks. The MP tries to run once every 80ms and, if accepted, produces a refined plan within 14ms.

The PND modeling the five tasks is depicted in figure 14. As in the previous example we use error places, named err_i with $i \in I = \{Lifter, Putter, TC, SR\}$, to model deadline misses. Unreachability of err_i implies liveness of the corresponding task. We have omitted these error places in the figure to avoid making it more complicated. For the same reason, we do not explicitly represent the construction used to enforce the 1-safety of the PND (in particular, of places P_1 and P_2). Compared to [13], our model requires stronger synchronization and imposes harder schedulability constraints as the buffer size can not exceed one.

The synthesis problem here consists in finding a scheduler that guarantees that no deadline is ever missed (i.e., $\Box\neg\bigvee_{i\in I} err_i$). Notice that the worst case CPU utilization factor (sum of the execution times divided by the periods) for the mandatory tasks (TC, Lifter, Putter, SR) is 0.817. This indicates that the scheduling constraints are really tight.

Table 2 shows the size of the synthesized deadlock-free scheduler[2] and the performance of the synthesis algorithm. Moreover, in the computed scheduler, the optional task MP can actually be scheduled in certain states.

5 Discussion

We have presented a framework for automatic scheduler generation based on a synthesis algorithm, and illustrated its applicability in practice to solve concrete examples.

Problem	TAD			Scheduler		Performances	
	states	trans	clks	states	trans	memory	time
2-process	17	32	4	16	28	3MB	4s
3-process	46	126	6	42	105	37MB	10m
robotic arm	349	1132	5	244	848	115MB	3h 20m

Table 2. Performances of the synthesis algorithm on an UltraSparc 5.

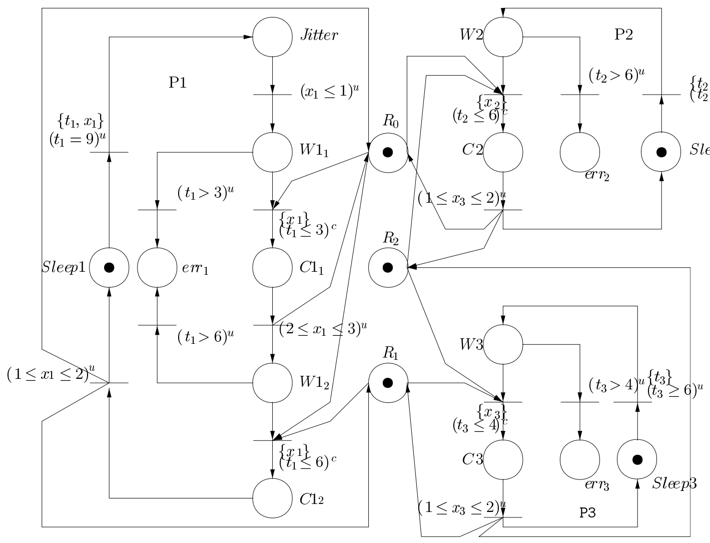


Figure 13. PND of the system with three tasks.

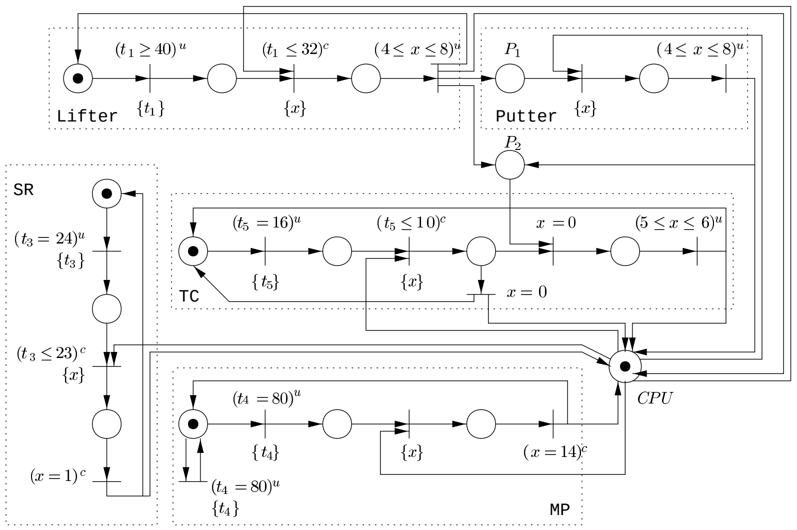


Figure 14. PND of the robotic arm.

We believe that the presented results are a basis for automatic scheduler generation of reactive applications that can be modeled as timed automata. They are not in principle applicable to scheduling with preemption, which requires models with integrators [10]. Compared to classical scheduling techniques (e.g. [12, 4]), our framework is applicable without any assumptions about the structure of the application such as periodicity or priorities of the tasks. Our algorithm is optimal in the sense that if it does not find a scheduler, then such a scheduler does not exist.

A limitation is certainly the theoretical complexity of the synthesis algorithm and of the generated schedulers. However, this complexity is not observed in the examples we have considered. We believe that the method is tractable for non-trivial systems of medium size. As the synthesized schedulers are maximal, that is, they contain all the schedules satisfying the given property, simpler deterministic schedulers could be obtained by reducing non-determinism.

Another way of avoiding state space explosion is to apply a compositional approach. The use of synchronization modes such as MAX and MIN, in addition to AND-synchronization, drastically helps keeping the discrete state complexity low.

We are currently studying both the possibility of defining direct composition rules and reducing non-determinism.

Related work. Our work is based on [14, 3]. Controller synthesis for timed automata has also been considered in [8], where the problem is reduced to the untimed framework of [16] using the *region graph* construction which results in state-space explosion. [15] treats the problem in the more general setting of linear hybrid automata, gives a semi-decision procedure (the problem is generally undecidable for this class of systems) based also on the symbolic fix-point algorithm of [14], and presents a prototype implementation in the tool HyTech. Due to the more general type of polyhedra used to represent symbolic states for hybrid automata, the operations are more costly.

The approach proposed in [11] is also similar to ours, in the sense that it uses an automata-based formalism [3], but it relies on a discrete-time semantic model and on a different algorithm based on the notion of weak-bisimulation. As far as we understand, it does not distinguish between controllable and uncontrollable transitions. We believe, however, that this distinction is fundamental to appropriately model the behavior of the environment.

To our knowledge, the KRONOS prototype that we have presented here is the first tool for controller synthesis specialized for timed automata.

References

[1] R. Alur and D.L. Dill. A theory of timed automata. *TCS*, 126, p. 183-235, 1994.

[2] K. Altisen. Génération automatique d'ordonnancements pour systèmes temporisés. Mémoire de DEA, ENSIMAG, 1998 (in french).

[3] E. Asarin, O. Maler, and A. Pnueli. Symbolic Controller Synthesis for Discrete and Timed Systems. In *Hybrid System II.* LNCS 999, Springer, 1995.

[4] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings. Deadline Monotonic Scheduling. In *8th Workshop on Real-time Operating Systems and Software.* IEEE, 1991.

[5] S. Bornot, J. Sifakis, and S. Tripakis. Modeling urgency in timed systems. In *COMPOS'97*, Malente, Germany. LNCS 1536, Springer-Verlag, 1998.

[6] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. Kronos: A model-checking tool for real-time systems. In *CAV'98.* LNCS 1427, Springer-Verlag, 1998.

[7] G. Gößler. Modélisation et contrôle des systèmes multimédia. Mémoire de DEA, ENSIMAG, 1998 (in french).

[8] G. Hoffmann and H. Wong Toi. The input-output control of real-time discrete event systems. In *30th IEEE Conf. on Decision and Control*, 1991.

[9] M. Jourdan, N. Layaïda, and L. Sabry-Ismail. Presentation Services in MADEUS: an Authoring Environment for Interactive Multimedia Documents. RR-2983, thème 3, INRIA, 1997.

[10] Y. Kesten, A. Pnueli, J. Sifakis, and S. Yovine. Integration graphs: A class of decidable hybrid systems. LNCS 736, Springer-Verlag. To appear in *Information and Computation.*

[11] H. Kwak, I. Lee, A. Philippou, J. Choi and O. Sokolsky. Symbolic schedulability analysis of real-time systems. In *RTSS'98*, Madrid, Spain, Dec. 1998. IEEE Computer Society Press.

[12] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1), 1973.

[13] M. Lusini and E. Vicario. Static analysis and dynamic steering of time-dependent systems using Petri Nets. Technical Report # 28.98, University of Florence, 1998.

[14] O. Maler, A. Pnueli, and J. Sifakis. On the Synthesis of Discrete Controllers for Timed Systems. In *STACS'95.* LNCS 900, Springer Verlag, 1995.

[15] H. Wong Toi. The synthesis of controllers for linear hybrid automata. Proc. of the *36th IEEE Conference on Decision and Control*, pp. 4607–4612, 1997.

[16] W. M. Wonham and P. J. Ramadge. On the supremal controllable sublanguage of a given language. *SIAM J. Control and Optimization*, 25(3):637–659, May 1987.

[3]after translation from the process algebra ACSR