Modeling Urgency in Timed Systems

Sébastien Bornot, Joseph Sifakis and Stavros Tripakis

VERIMAG
Centre Équation, 2, rue de Vignate, 38610, Gières, France
E-mail: {bornot,sifakis,tripakis}@imag.fr

1 Introduction

Timed systems can be modeled as automata (or, generally, discrete transition structures) extended with real-valued variables (*clocks*) measuring the time elapsed since their initialization. The following features are also common in the above models.

- States are associated with *time progress conditions* specifying how time can advance. Time can progress at a state by t only if all the intermediate states reached satisfy the associated time progress condition.
- At transitions, clock values can be tested and modified. This is usually done by associating with transitions guards (conditions on clocks) and assignments. If a guard is true from an automaton state and a given clock valuation, the corresponding transition can be executed by modifying clocks as specified by the corresponding assignment.

Time progress conditions can be used to specify urgency of transitions. Maximal urgency is achieved at a state if the corresponding time progress condition is equal to the negation of the disjunction of the guards of the transitions issued from this state. This implies that waiting at the state is allowed only if there is no enabled transition. As soon as a transition is enabled, time cannot progress anymore and the execution of the enabled transition(s) is enforced. Minimal urgency is achieved at a state when the corresponding time progress condition is true which implies that time can advance forever from this state and consequently indefinite waiting is allowed.

Choosing appropriate time progress conditions for complex system specifications is not a trivial problem as it is claimed in [SY96,BS97b,BS97a]. In many papers, time progress conditions have been defined as *invariants* that must be continuously true by clock valuations at the corresponding states. This implies that when a state is reached the associated invariant must be satisfied and makes modeling of absolute urgency sometimes difficult (for instance, in the case where a transition must be executed as soon as it is enabled).

The problem of the definition and use of time progress conditions has been tackled in [SY96,BS97b]. The purpose of this work is to show how the application of results presented in [BS97b] leads to a modeling methodology for timed systems. Emphasis is put on pragmatic and methodological issues. The basic ideas are the following.

- A timed system can be specified as the composition of *timed transitions*. The latter are transitions labeled, as usual, with guards and assignments but also with deadlines, conditions on the clocks that characterize the states at which the transition is enforced by stopping time progress. We require that the deadline of a transition implies its guard, so that whenever time progress is stopped the transition is enabled.
- The guards and deadlines may contain formulas with past and future modalities concerning the evolution of clock values at a state. The use of such modalities does not increase the expressive power of the model but drastically enhances comfort in specification.

The paper is organized as follows. In section 2 we define Timed Automata with Deadlines (TAD) which are a class of Timed Automata [ACD93,HNSY94] where time progress conditions depend on deadlines associated with transitions. We show that using TAD makes urgency specification easier. In section 3 we present the model of Petri Nets with Deadlines (PND), which are (1-safe) Petri nets extended with clocks exactly as TAD are extensions of automata. We compare PND with different classes of Timed Petri Nets (TPNs) and show that safe TPNs can be modeled as PND. Section 4 presents some applications to modeling systems and in particular to modeling multimedia documents.

2 Timed Automata with Deadlines

2.1 Definitions

Definition 1 *(Timed Automaton with Deadlines (TAD))*
A TAD is :

- A discrete labeled transition system $(S, \rightarrow, A)$ where
  - S is a finite set of discrete states
  - A is a finite vocabulary of actions
  - $\rightarrow \subseteq S \times A \times S$ is a untimed transition relation
- A set $X = \{x_1, \dots, x_m\}$ of real-valued variables called *clocks* with $dom(x_i) \in \mathbf{R}_+$.
- A labeling function h mapping *untimed transitions*, elements of $\rightarrow$, into *timed transitions*: $h(s, a, s') = (s, (a, g, d, r), s')$, where
  - g, d are respectively the *guard* and the *deadline* of the transition. Guards and deadlines are predicates p defined by the following grammar :

    $$p ::= x\#c \mid x - y\#c \mid p \wedge p \mid \neg p$$

    where $x, y \in X$, c is an integer and $\# \in \{\leq, <\}$. We assume that $d \Rightarrow g$.
  - $r \subseteq X$ is a set of clocks to be reset.

Definition 2 *(Semantics of a TAD)*
A *state* of a TAD is a pair (s, v), where $s \in S$ is a discrete state and $v \in \mathbf{R}_+^m$

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

Given $s \in S$, if $\{(s, a_i, s_i)\}_{i \in I}$ is the set of all the transitions issued from s and $h(s, a_i, s_i) = (s, (a_i, g_i, d_i, r_i), si)$ then :

- $\forall i \in I\ \forall v \in \mathbf{R}_+ \ .\ (s, v) \xrightarrow{a_i} (s_i, v[r_i])$ if $g_i(v)$ where $v[r_i]$ is the variable valuation obtained from v when all the clocks in r_i are set at zero (and the others left unchanged).
- $(s, v) \xrightarrow{t} (s, v + t)$ if $\forall t' < t \ .\ c_s(v + t')$ where $c_s = \neg \bigvee_{i \in I} d_i$ and $v + t$ is the valuation obtained from v by increasing all the clock values by t.
  We call c_s the *Time Progress Condition* (TPC) associated with the discrete state s.

We consider TAD such that for any state s the TPC c_s is right-open.

2.2 About time-progress conditions

Notice that the simplest TAD is a single timed transition $(s, (a, g, d, r), s')$ with untimed transition (s, a, s'), guard g, deadline d and reset set r. The guard g characterizes the set of states from which the timed transition is possible while the deadline d characterizes the subset of these states where the timed transition is enforced by stopping time progress. The relative position of d with respect to g determines the *urgency* of the action. For a given g, the corresponding d may take two extreme values: first, $d = g$, meaning that the action is *eager* and, second, $d = false$, meaning that the action is *lazy*. A particularly interesting case is the one of a *delayable* action where d is the *falling edge* of a right-closed guard g (cannot be disabled without enforcing its execution). The above cases are illustrated in figure 1.

The condition $d \Rightarrow g$ guarantees that if time cannot progress at some state, then at least one action is enabled from this state. Restriction to right-open TPCs guarantees that deadlines can be reached by continuous time trajectories and permits to avoid deadlock situations in the case of eager transitions. (For instance, consider the case where $d = g = x > 2$, implying the TPC $x \leq 2$, which is not right-open. Then, if x is initially 2, time cannot progress by any delay t, according to definition 2.1 above. The guard g is not satisfied either, thus, the system is deadlocked.) The assumptions above ensure the property of *time reactivity*, that is, time can progress at any state unless a untimed transition is enabled.

Branching from a state s can be considered as a non-deterministic choice operator between all the timed transitions issued from this state. The resulting untimed transition relation is the union of the untimed transition relations of the combined timed transitions. The resulting time step relation is the intersection of the time step relations of the combined timed transitions.

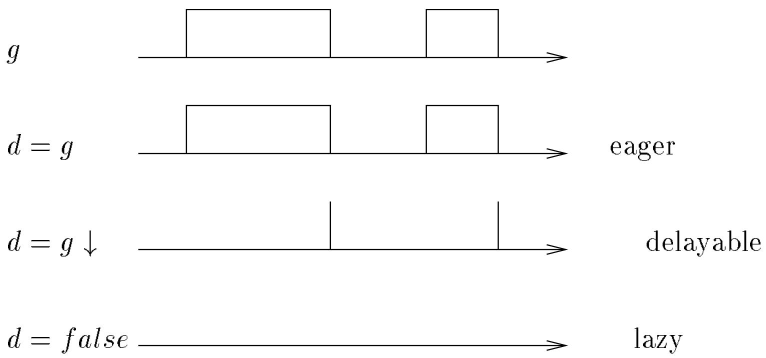


Fig. 1. Using deadlines to specify urgency.

Compared to the Timed Automata (TA) model [HNSY94], TAD differ in that TPCs are not given explicitly but rather derived from the deadlines which specify urgency of individual timed transitions. Thus, TAD are a subclass of TA that are time-reactive.

We believe that using deadlines rather than directly TPCs allows an easier modeling of urgency. Consider, for example, the TA in figure 2 which differ only in their TPCs. Clearly, the TA (1) and (2) specify the same behavior when s is reached with values $x \leq 5$. However, (1) does not satisfy the time reactivity requirement and cannot be obtained from a TAD, while (2) can be obtained by supposing that a is delayable ($d_1 = 5$) and b is eager or delayable ($d_2 = 5$). The case (3) corresponds to eager actions a and b and (4) to lazy actions.

Definition 3 *(Urgency types)*
For convenience, we replace explicit deadlines in transitions by the *urgency types* $\wr\!\downarrow$, $\downarrow$, $\wr$, which are simply notations meaning that a transition is eager ($d = g$), delayable ($d = g\downarrow$), lazy ($d = false$), respectively.

Notice that any TAD can be transformed into an equivalent TAD with only eager and lazy transitions.

For complex systems, computation of TPCs from deadlines of transitions may be useful as shown by the following example. In table 2.2 we give the TPCs c_s associated with state s (figure 3) for different types of urgency ($\wr\!\downarrow$ = eager, $\downarrow$ = delayable, $\wr$ = lazy) of the transitions (s, a_1, s_1) and (s, a_2, s_2).

δ_2 \ δ_1	$\wr$	$\downarrow$	$\wr\!\downarrow$
$\wr$	$false$	$x = 5$	$2 \leq x \leq 5$
$\downarrow$	$y = 7$	$y = 7 \vee x = 5$	$y = 7 \vee 2 \leq x \leq 5$
$\wr\!\downarrow$	$4 \leq y \leq 7$	$4 \leq y \leq 7 \vee x = 5$	$4 \leq y \leq 7 \vee 2 \leq x \leq 5$

Notice that the use of urgency types to induce deadlines could lead to right-closed TPCs (for example, consider the case where a transition is eager and has

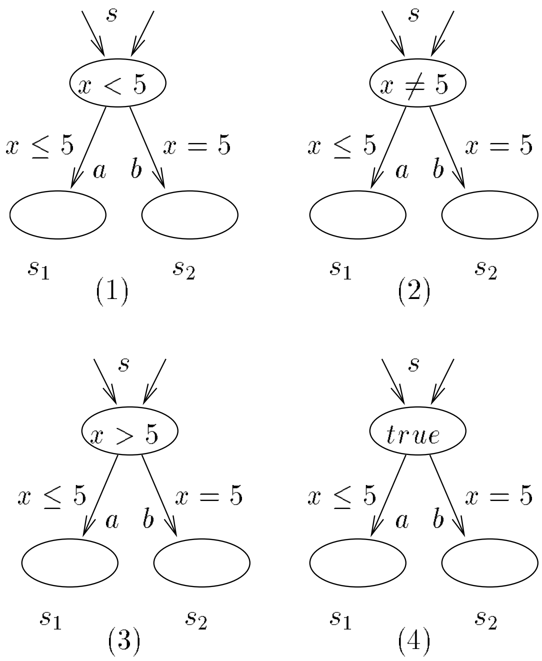


Fig. 2. Modeling urgency with TPCs.

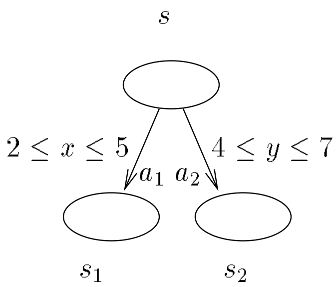


Fig. 3. Computing TPCs.

a left-open guard, say, $1 < x < 2$). This can be avoided by ensuring that eager transitions have always left-closed guards.

2.3 Priority Choice

It is often useful to consider that some priority is applied when from a state several timed transitions are enabled. This amounts to taking the non-deterministic choice between the considered transitions by adequately restricting the guards of the transitions with lower priority.

Consider, for example, two timed transitions $(s, (a_i, g_i, d_i, r_i), s_i)$ for $i = 1, 2$ with a common source state s. If a_1 has lower priority than a_2 in the resulting TAD the transition labeled by a_2 does not change while the transition labeled by a_1 becomes $(s, (a_1, {g'}_1, {d'}_1, r_1), s_1)$ where ${g'}_1 \Rightarrow g_1$ and ${d'}_1 = d_1 \wedge {g'}_1$.

Commonly, g'_1 is taken to be $g_1 \wedge \neg g_2$, which means that whenever a_1 and a_2 are simultaneously enabled, a_1 is disabled in the prioritized choice. However, for timed systems other ways to define g'_1 are possible. One may want to prevent action a_1 to be executed if it is established that a_2 will be eventually executed within a given delay.

For this reason we need the following notations.

Definition 4 *(Modal operators)*
Given a predicate p on X as in definition 2.1, we define the modal operators $\Diamond_{\leq k}\, p$ ("eventually p within k") and $\overleftarrow{\Diamond}_{\leq k}\, p$ ("once p since k"), for $k \in \mathbf{R}_+ \cup \{\infty\}$.

$$\Diamond_{\leq k}\ p\ (v) \text{ if } \exists t \in \mathbf{R}_+\ 0 \leq t \leq k.\ p(v+t)$$
$$\overleftarrow{\Diamond}_{\leq k}\ p\ (v) \text{ if } \exists t \in \mathbf{R}_+\ 0 \leq t \leq k.\ \exists v' \in V.\ v = v' + t \wedge p(v')$$

We write $\Diamond p$ and $\overleftarrow{\Diamond}\, p$ for $\Diamond_{\leq\infty}\, p$ and $\overleftarrow{\Diamond}_{\leq\infty}\, p$, respectively, and $\Box p$ and $\overleftarrow{\Box}\, p$ for $\neg\Diamond\neg p$ and $\neg\overleftarrow{\Diamond}\, \neg p$, respectively.

Notice that modalities can be eliminated to obtain simple predicates without quantifiers. For example, $\Diamond(1 \leq x \leq 2)$ is equivalent to $x \leq 2$. For notational convenience, we shall be using in the sequel guards and deadlines with modalities.

Coming back to the example above, we can take $g'_1 = g_1 \wedge \neg\Diamond_{\leq k} g_2$ or even $g'_1 = g_1 \wedge \Box\neg g_2$. In the former case, a_1 gives priority up to a_2 if a_2 is eventually enabled within k time units. In the latter case, a_1 is enabled if a_2 is disabled forever.

It is shown in [BS97b] that for timed systems it is possible to define priority choice operators applicable to a set of timed transitions and parameterized by a priority relation $< \subseteq A \times \mathbf{R}_+ \times A$. If $(a_1, k, a_2) \in <$ (denoted $a_1 <_k a_2$) then the priority choice applied to a given set of timed transitions restricts the guard g_1 of a transition labeled by a_1 so as to disable a_1 whenever a_2 is to be enabled within k time units. In [BS97b] is is also shown that if the priority order satisfies some "transitivity conditions" then the corresponding priority choice preserves deadlock freedom in the following sense: If $\{g_i\}_{i \in I}$ are the guards of a set of timed transitions and $\{g'_i\}_{i \in I}$ are the modified guards obtained by application of the priority-choice operator then $\Diamond \bigvee_{i \in I} g_i \equiv \Diamond \bigvee_{i \in I} g'_i$ and $\Diamond g_i \Rightarrow \Diamond(g'_i \vee$

$\bigvee_{\exists k . a_i <_k a_j} g'_j)$. The latter property says that if from a state the i-th transition is eventually enabled in the non-deterministic choice, then in the prioritized choice, either the i-th transition will be eventually enabled, or some transition of higher priority.

Let us illustrate the above ideas with an example. Consider the priority choice between two timed transitions with respective labels (a_i, g_i, d_i, r_i), $i = 1, 2$, such that a_1 has lower priority than a_2, where $g_1 = 0 \leq x \leq 4 \vee x \geq 6$ and $g_2 = 2 \leq x \leq 7$ for some x. We get the following decreasing values for ${g'}_1$ as the priority delay increases:

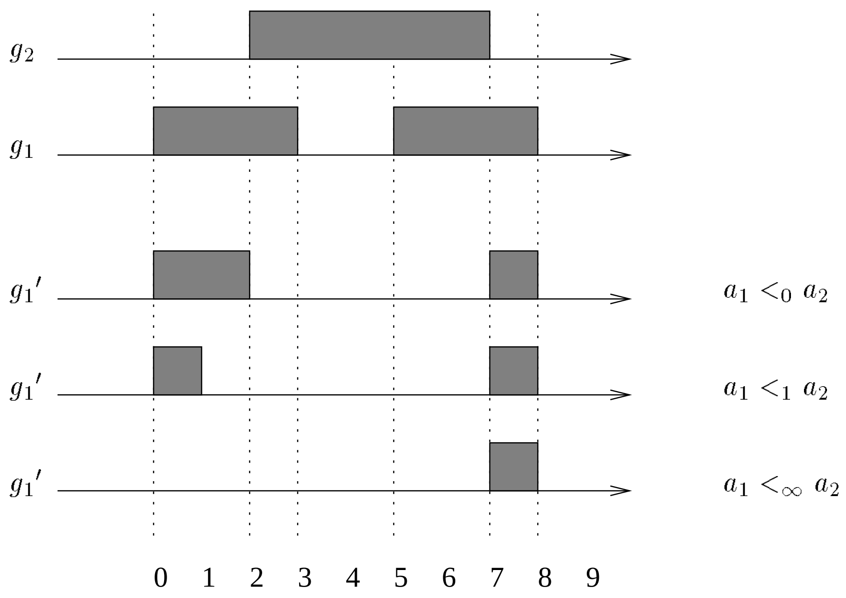


Fig. 4. Different priorities for a_2 over a_1.

$$
\begin{array}{ll}
g'_1 = g_1 \wedge \neg g_2 = 0 \leq x < 2 \vee x > 7 & \text{(immediate priority)} \\
g'_1 = g_1 \wedge \neg \Diamond_{\leq 1} g_2 = 0 \leq x < 1 \vee x > 7 & \text{(priority within a delay of 1)} \\
g'_1 = g_1 \wedge \neg \Diamond g_2 = x > 7 & \text{(priority within an infinite delay)}
\end{array}
$$

Figure 4 illustrates the above example. The first case corresponds to the "classical" priority choice, where a_1 is disabled whenever a_2 is enabled. The second case is stronger: a_1 is disabled also in case a_2 becomes enabled in at most 1 time unit. The third case is the strongest: a_1 is disabled whenever it is possible for a_2 to become enabled sometime in the future.

Finally, we should note that the use of negations to generate priority could lead to right-closed TPCs. When urgency types are used, this can be avoided by ensuring that a lazy transition never has higher priority over an eager transition.

3 Petri Nets with Deadlines

3.1 Definition

For the sake of simplicity, we consider the timed extensions of *1-safe* Petri nets.
Definition 5 *(Petri Net with Deadlines (PND))*
A PND consists of :

- 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.
  - $\mathcal{T} \subseteq 2^{\mathcal{P}} \times A \times 2^{\mathcal{P}}$ is a transition relation.
- A set $X = \{x_1, \ldots, x_m\}$ of clocks.
- A labeling function h mapping untimed transitions elements of $\mathcal{T}$ into timed transitions : $h(P, a, P') = (P, (a, g, d, r), P')$, where $P, P' \subseteq \mathcal{P}$.

As usually, we represent a PND as a bipartite labeled graph with two types of nodes (places and transitions), see figure 5. The transitions are labeled with action names, guards, deadlines and resets.

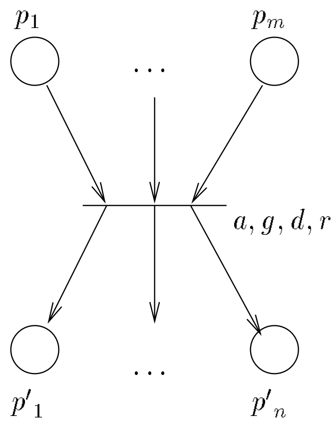


Fig. 5. The transition $(\{p_1, \ldots, p_m\}, (a, g, d, r), \{p'_1, \ldots, p'_n\})$.

We define the semantics of a PND in terms of a TAD.
Definition 6 *(TAD associated to a PND)*
A PND $(\mathcal{P}, \mathcal{T}, A, X, h)$ defines a TAD $(S, \rightarrow, A, X, h')$ such that :

- $S = 2^{\mathcal{P}}$
- $P \xrightarrow{a} P'$ if $(P, a, P') \in \mathcal{T}$
- $h'(P, a, P') = h(P, a, P')$.

The above definition simply means that a PND is a TAD where the discrete transition structure is the corresponding marking graph. The transitions of the marking graph are submitted to the same timing constraints as the transitions of the PND. So PND are extensions of PNs where transitions are submitted to timing constraints exactly as TAD are extension of automata.

By adopting standard PN terminology, we will say that there is a token in place p when p is an element of the current state in the marking graph. Places are local states of processes. A transition with several input places represents a synchronization of several processes. It is enabled only if its input places have a token and the associated timing constraints are satisfied.

An example of PND is given in figure 6.

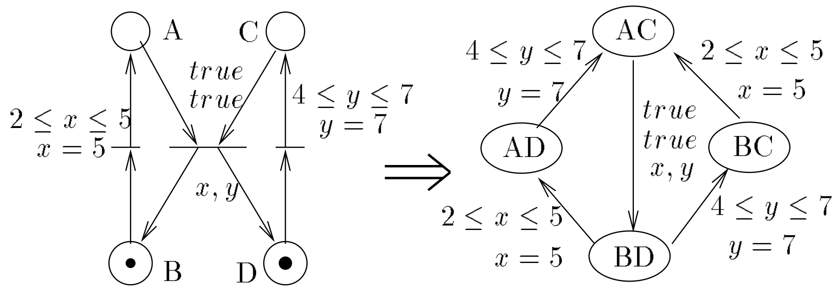


Fig. 6. A PND and its corresponding TAD.

3.2 Synchronization modes

We introduce some useful macro-notations that allow concise description of synchronization guards in terms of timing constraints about the synchronizing processes.

We first define three different synchronizing modes that correspond to different types of coordinations between processes. We suppose that, for a synchronization transition, are given "local guards" g_i expressing timing constraints about termination of each contributing process. We associate each guard g_i with an input arc of the synchronization transition (figure 7). A mode defines a way of composing the guards g_i to obtain the synchronization guard g.

AND-synchronization : The resulting guard g is the conjunction $g = \bigwedge_{i \in [1 \ldots n]} g_i$ of the input guards. This simply means that synchronization is possible only if all processes can terminate together. In the example of figure 8, we get $g = g_1 \wedge g_2 = 3 \leq x \leq 7$.

MAX-synchronization : Synchronization can take place only if all the contributing processes have terminated. This implies synchronization at times t

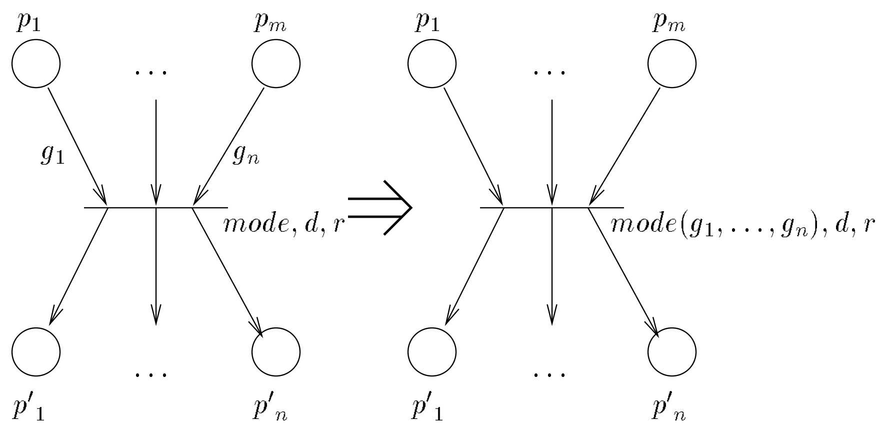


Fig. 7. Meaning of synchronization notation.

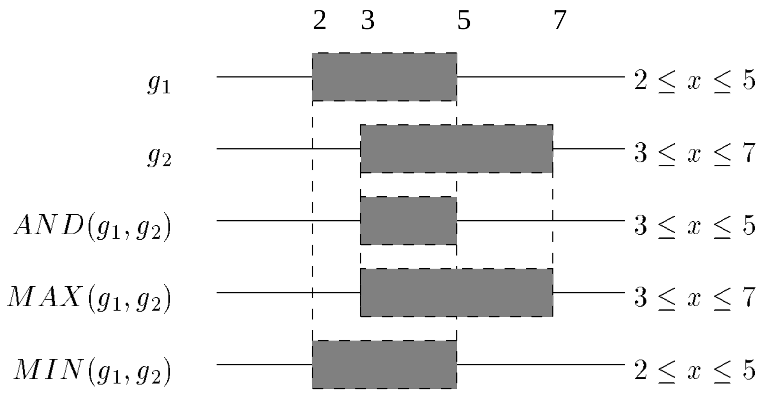


Fig. 8. Resulting guards for the three synchronization modes.

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 $g = \bigvee_{i \in [1 \ldots n]} g_i \wedge \bigwedge_{j \neq i} \Diamond^{-} g_j$. The i-th term of the guard means that the i-th process can terminate (now) while the others have already terminated. This allows to specify synchronization with mutual waiting of all the contributing processes if $\Diamond g_i$ holds when the input place p_i is reached. Otherwise it may happen that before reaching an input place p_i the guard has been already satisfied but this does not correspond to termination of p_i.

In the example of figure 8, we get $g = g_1 \wedge (\Diamond^{-} g_2) \vee (\Diamond^{-} g_1) \wedge g_2 = 3 \leq x \leq 7$.

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 $g = \bigvee_{i \in [1 \ldots n]} g_i \wedge \bigwedge_{j \neq i} \Diamond g_j$. The i-th term of the guard means that the i-th process can terminate (now) and all the others will eventually terminate.

For the example of figure 8, we get $g = g_1 \wedge \Diamond g_2 \vee \Diamond g_1 \wedge g_2 = 2 \leq x \leq 5$.

3.3 Translating safe timed Petri nets into PND

Many different classes of timed Petri nets (TPNs) have been defined. An important difference between TPNs and PND is that in the former timing constraints are local and associated with tokens. A comparison of the two models in the general case of non-safe Petri nets is out of the scope of this paper and is the object of an ongoing work. Here, we restrict our attention to 1-safe TPNs.

Place-TPNs : [Sif77] In this class of TPNs, intervals $[l_i, u_i]$ are associated with places p_i. A token arriving at a place p_i cannot be used for firing an output transition for some time t, $l_i \leq t \leq u_i$. After this time it becomes available. A transition fires as soon as all its input places have available tokens.

The principle of a method for translating Place-TPNs to PND is illustrated in figure 9.

Transition-TPNs : [Mer74] In this class of TPNs, intervals $[l_i, u_i]$ are associated with transitions τ_i. A timed transition τ_i fires in times t, $l_i \leq t \leq u_i$ after the corresponding untimed transition becomes enabled.

The principle of a method for translating Transition-TPNs to PND is illustrated in figure 10.

Stream-TPNs : This type of TPNs is introduced in [SDdSS94]. Given a transition τ, an interval $[l_i, u_i]$ is associated with each input arc (p_i, τ) of τ. A token entering the input place p_i must wait for a time t, $l_i \leq t \leq u_i$, before becoming available for the transition τ.

Nine different synchronization modes for stream TPNs are defined in [SDdSS94] (see figure 11). For each input place p_i of the synchronization transition τ, two timers x_i and y_i are defined as follows: $x_i \stackrel{\text{def}}{=} max(l_i - t_i, 0)$

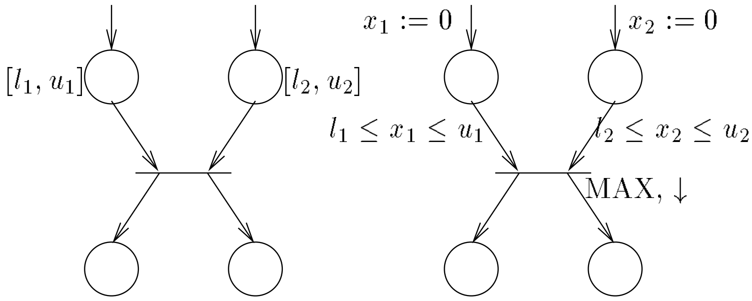


Fig. 9. From Place-TPNs to PND.

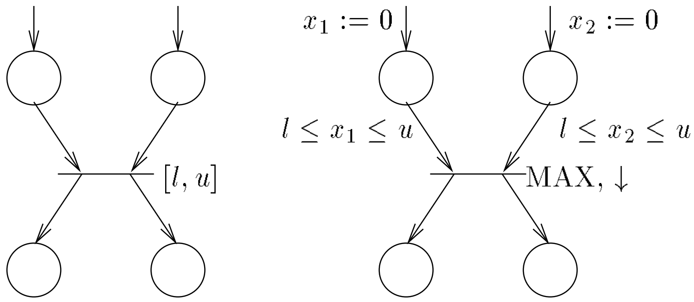


Fig. 10. From Transition-TPN to PND.

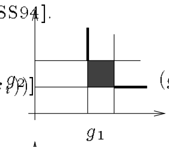
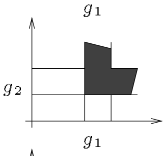
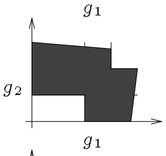
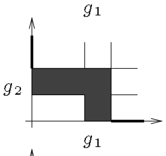
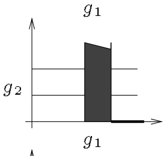
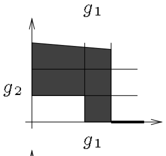
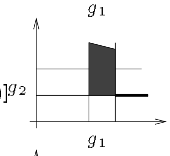
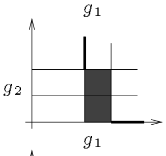
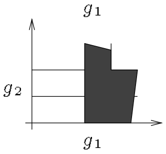
Synchronization modes [SDdSS94].	Schematically, for two components.	The corresponding TAD guard.
AND $[max(x_i), max(min(y_i), max(x_i))]$		$(g_1 \wedge g_2) \vee ((g_1 \wedge \neg\Diamond g_2)\downarrow) \vee ((g_2 \wedge \neg\Diamond g_1)\downarrow)$
WEAK-AND $[max(x_i), max(y_i)]$		$(g_1 \wedge \Diamond g_2) \vee (g_2 \wedge \Diamond g_1)$
OR $[min(x_i), max(y_i)]$		$(\Diamond g_1 \wedge \Diamond g_2) \vee (\Diamond g_2 \wedge \Diamond g_1)$
STRONG-OR $[min(x_i), min(y_i)]$		$(g_1 \wedge \Diamond g_2) \vee (g_2 \wedge \Diamond g_1)$ $\vee$ $((\neg\Diamond g_1)\downarrow) \vee ((\neg\Diamond g_2)\downarrow)$
MASTER $[x_m, y_m]$		$g_1 \vee ((\neg\Diamond g_1)\downarrow)$
OR-MASTER $[min(x_i), y_m]$		$g_1 \vee ((\neg\Diamond g_1)\downarrow) \vee (\Diamond g_1 \wedge \Diamond g_2)$
AND-MASTER $[max(x_i), max(y_m, max(x_i))]$		$(g_1 \wedge \Diamond g_2) \vee ((g_2 \wedge \neg\Diamond g_1)\downarrow)$
STRONG-MASTER $[x_m, max(min(y_i), x_m)]$		$(g_1 \wedge \Diamond g_2) \vee ((g_1 \wedge \neg\Diamond g_2)\downarrow)$
WEAK-MASTER $[x_m, max(y_i)]$		$g_1 \vee (\Diamond g_1 \wedge \Diamond g_2)$

Fig. 11. Synchronization mode for Stream-TPNs.

and $y_i \stackrel{\text{def}}{=} max(u_i - t_i, 0)$, where t_i is the elapsed time since the arrival of the token at place p_i. Thus, x_i and y_i are taken to be the "current" lower and upper bounds for the enabledness of each input arc (p_i, τ).

The nine synchronization modes are shown in the leftmost column of figure 11. Notice that, in the figure, max and min denote the usual mathematical operators and are not to be confused with the MAX and MIN synchronization operators defined previously. Also, we write $max(x_i)$ as a shorthand for $max_{i=1,\ldots n}\{x_i\}$, and similarly for min. The middle column of the figure displays the guards induced by each of the synchronization modes, for $n = 2$, where $g_i = [l_i, u_i]$.

The model of stream-TPNs can also be translated into our model, as shown in the right-most column of figure 11.

4 Applications

4.1 Producer – Consumer

We show how PND can be used to model a system composed of a producer and a consumer communicating via a zero-length buffer. The producer takes between l_p and u_p time units to produce an item, which is then made available to the buffer after a delay between l'_p and u'_p time units. The consumer needs between l_c and u_c time units to consume an item and is ready for a new item after a delay between l'_c and u'_c time units. The above delays are measured using one clock per process, namely, x for the producer and y for the consumer. Figure 12(a) shows the two processes modeled as PND.

Whenever the buffer is full and the consumer is willing to take an item, the latter is exchanged between the two processes by an instantaneous *handshake*. The latter is represented by the synchronization transition of the PND corresponding to the composition of the two processes, shown in figure 12(b). The guard g of the handshake transition can be chosen to be either g' or g'', where:

$$\begin{aligned} g' &\equiv \mathrm{AND}(l'_p \leq x \leq u'_p, l'_c \leq y \leq u'_c) \\ &\equiv l'_p \leq x \leq u'_p \ \wedge \ l'_c \leq y \leq u'_c \\ g'' &\equiv \mathrm{MAX}(l'_p \leq x \leq u'_p, l'_c \leq y \leq u'_c) \\ &\equiv l'_p \leq x \leq u'_p \wedge l'_c \leq y \ \vee \ l'_p \leq x \wedge l'_c \leq y \leq u'_c \end{aligned}$$

In the first case, the temporal constraints are considered "hard", that is, it is required that both lower and upper bounds of the intervals $[l'_p, u'_p]$ and $[l'_c, u'_c]$ are respected in order for the handshake to take place. (An informal explanation of this choice could be that u'_p represents the "expiring date" of the item, while u'_c is the maximum time the consumer can wait, after which he/she "starves to death".) AND synchronization is commonly used in the composition of systems, however, it is a strict synchronization mechanism which often leads to deadlocks.

In the case of MAX synchronization, temporal constraints are "looser", that is, only one of the upper bounds is required to hold. (Informally, this might represent a more realistic situation, where the item never looses its value, while

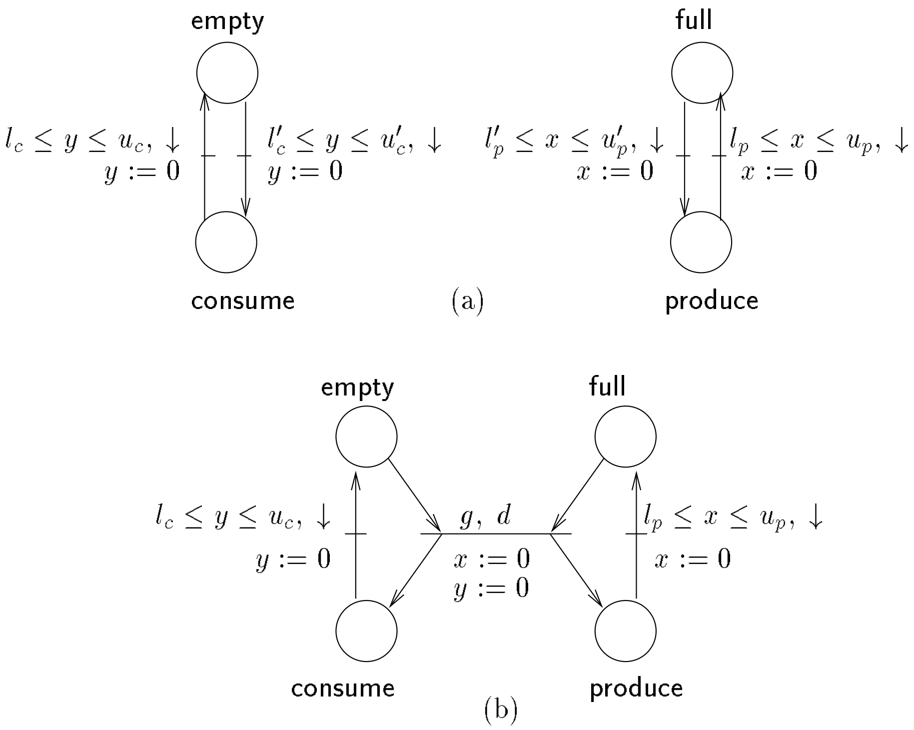


Fig. 12. Producer–Consumer system modeled as PND.

the consumer is willing to wait.) MAX synchronization guarantees the absence of deadlocks. Moreover, combined with appropriate deadlines, it can model synchronization with minimal or maximal waiting, as we show below.

Regarding the urgency type of the synchronization transition, in the case of AND synchronization it is reasonable to assume that the transition is delayable, which gives the deadline:

$$d' \;\equiv\; g' \downarrow \;\equiv\; x = u'_p \wedge l'_c \leq y \leq u'_c \;\vee\; y = u'_c \wedge l'_p \leq x \leq u'_p$$

In the case of MAX synchronization more than one possibilities are of interest, namely:

- The choice of delayble transition corresponds to a maximal-waiting policy:

$$d'' \;\equiv\; g'' \downarrow \;\equiv\; x = u'_p \wedge y \geq u'_c \;\vee\; y = u'_c \wedge x \geq u'_p;$$

- The choice of eager transition corresponds to a minimal-waiting policy;
- The following choice corresponds to a "best-effort" synchronization scheme, where either no upper bound is violated if possible, or the transition is executed as soon as possible, in the case of violation:

$$d''' \;\equiv\; d' \;\vee\; x = l'_p \wedge y \geq u'_c \;\vee\; y = l'_c \wedge x \geq u'_p.$$

Notice that d''' cannot be obtained using any of the urgency types $\wr\!\!\downarrow$, $\downarrow$ or $\wr$.

4.2 Variations on the theme of mutual exclusion

We consider the generic mutual-exclusion situation shown in figure 13. A resource is shared by two processes P_1 and $P2$ and can be used by at most one of them at any time. Each time it is used, the resource is again available after an amount of time which can vary in an interval I. Process P_i occupies the resource for an amount of time in an interval C_i, for $i = 1, 2$. From the moment it has finished using the resource, P_i is ready to use it again after some delay in an interval W_i. In the PND model shown in the figure, clocks x_1, x_2 and z are used for P_1, P_2 and the resource, respectively.

There are different policies of granting the resource to the processes, depending on how strict the temporal constraints of the problem are taken to be and also on whether an optimal utilization of the resource is sought. We examine some of these policies below, showing how they can be modeled by appropriately choosing the guards g_i and the urgency types δ_i shown in the figure, for $i = 1, 2$. We assume that $I = [l, u]$, $W_i = [l_i, u_i]$ and $C_i = [l'_i, u'_i]$, for $i = 1, 2$ (the analysis can be generalized to unbounded intervals).

- $g_i \equiv \mathrm{AND}(x_i \in W_i, z \in I)$. In this case the temporal constraints are hard. Then, if process P_i manages to get the resource, it is guaranteed to do so at most u_i time units after the time it has released it. On the other hand, the resource is guaranteed not to be left idle for more than u time units after it has been used for the last time. The problem of this method is that it can easily lead to deadlocks, either local (i.e., where one process starves) or global (i.e., where the whole system is blocked).

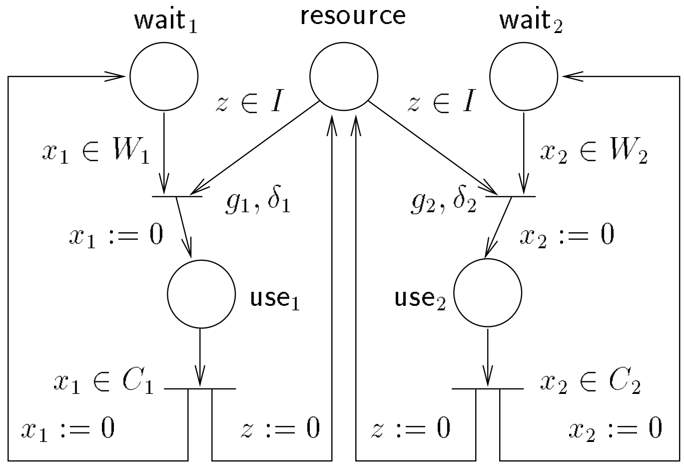


Fig. 13. Mutual exclusion modeled as PND.

- $g_i \equiv \mathrm{MAX}(x_i \in W_i, z \in I)$. In this case the temporal constraints are loose and the specification is deadlock-free for any (non-empty) intervals I, W_i and C_i.
- $g_i \equiv (x_i \in W_i) \wedge \diamondsuit\, (z \in I)$ or $g_i \equiv (\diamondsuit\, x_i \in W_i) \wedge (z \in I)$. These are intermediate choices, looser than AND synchronization, however, without avoiding deadlocks completely. In the first case, the upper bound of the resource's interval is ignored, while in the second case, the processes' upper bounds are ignored.

Regarding the urgency type of the synchronization transition, it can be chosen to be either eager or delayable (lazy synchronization is not meaningful in this case). Delayable is the less strict choice, minimizing the risk of deadlocks in the case MAX is not used. Eager implies that a better utilization of the resource (i.e., less idle time) is achieved. However, if MAX synchronization is not used, the risk of deadlocks is greater than in the delayable case, since the time non-determinism is reduced.

We finally consider the situation where process P_1 is given a higher priority with respect to process P_2. This is typically the case when P_1 demands the resource much less frequently than P_2 (for example, when P_1 is the process handling the keyboard, while P_2 is any batch process). We can model the different priorities by enforcing the guard g_2 into $g_2' = g_2 \wedge \neg \diamondsuit_{\leq u_2'} g_1$, where u_2' is the upper bound of interval C_2. The intention is to let P_2 have the resource only if it is guaranteed to finish before P_1 becomes ready.

4.3 Deadline-monotonic scheduling without preemption

We consider the following real-time scheduling problem. We are given a single processor and a set of *periodic* tasks $P_1, \ldots, P_n$ to be executed upon this processor.

Task P_i has a computation delay C_i and becomes ready for execution every T_i time units (the *period* of P_i). Furthermore, P_i needs to be completed at most D_i time units after the moment it becomes ready (the *deadline* of P_i). We assume that, for each $i = 1, ..., n$, we have $C_i \leq D_i \leq T_i$. The processor can execute only one process at a time and no *preemption* is allowed, that is, execution of a process cannot be interrupted and continued later on. See figure 14.

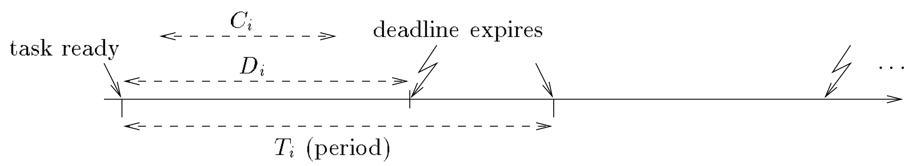


Fig. 14. Deadline-monotonic scheduling assumptions.

We show how Petri nets with deadlines can be used to model the so-called *deadline-monotonic* algorithm [ABRW91] which solves the above scheduling problem. [1] The algorithm is based on assigning *static priorities* to tasks according to their deadlines. In particular, higher priorities are assigned to tasks with shorter deadlines and no two tasks have the same priority (in case two tasks have equal deadlines, their relative ordering is chosen arbitrarily).

Figure 15(a) shows the PND modeling task P_i. The net has three places, namely, sleep_i (the task hasn't become ready yet), wait_i (the task is ready and waiting to be served) and use_i (the task is being served). Two clocks are used per task, namely, x_i and y_i: x_i counts the period T_i and also makes sure that the deadline D_i is not violated; y_i counts the computation delay C_i.

Figure 15(b) shows the PND modeling the deadline-monotonic scheduling algorithm for two tasks P_1 and P_2, assuming that the first one has higher priority (i.e., $D_1 \leq D_2$). The processor is modeled as a single place the token of which is necessary in order for a task to execute. Priority of P_1 over P_2 is ensured by placing the guard $x_1 < T_1$ in the transition $\mathsf{wait}_2 \rightarrow \mathsf{use}_2$. Transitions $\mathsf{wait}_i \rightarrow \mathsf{use}_i$ are both eager while all other transitions are delayable.

Using KRONOS, we test the schedulability of two tasks for various values of the parameters $C_i, D_i, T_i, i = 1, 2$. The test is performed as follows. We first replace the parameters by their values and generate the TAD corresponding to the resulting PND. Next, we translate this TAD to a classical TA with time-progress conditions by using extra clocks to specify the urgency of certain transitions. Finally, we test whether in the TA there exist reachable states which are zeno, that is, from which time can no longer progress. In fact, there are two cases: either all reachable states of this TA are zeno, meaning that the tasks are not schedu-

[1] We model a simplified version of the algorithm. Actually, deadline-monotonic scheduling uses preemption.

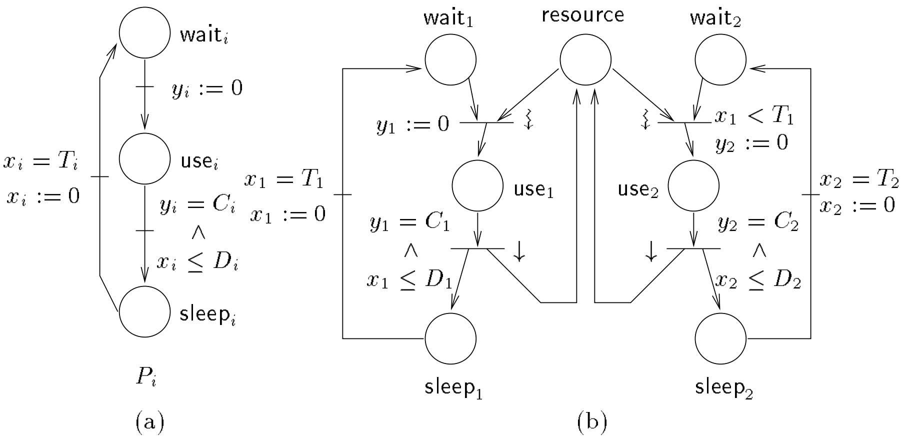


Fig. 15. Deadline-monotonic scheduling modeled as PND.

lable, or no zeno reachable states exist, which means that deadline-monotonic scheduling can be applied.

4.4 Specification and verification of multimedia documents

Description This application deals with modeling a multimedia document as a PND which can then be analyzed in order to check whether the document admits an execution scenario. More precisely, we consider (a simplified version of) MADEUS [JLSIR97] as the specification language of multimedia documents. This language combines operators from Allen's *interval temporal logic* [All83] with waiting and interruption operators.

The building blocks of a document are *media objects* representing a piece of information which has to be "played" continuously for a certain duration. The latter can be either fixed, or variable, in which case some flexibility is allowed in the presentation of the object. Let $\mathcal{O} = \{O_1, ..., O_n\}$ be the set of media objects. With each O_i we associate a *duration interval* I_i of one of the following types: $[l, u]$, $[l, u)$, $[l, \infty)$ or (l, ∞), where l, u are natural constants.

Documents are tree-like structures, built according to the following syntax:

$$\mathcal{D} ::= O \mid \mathcal{D}_1 \ \mathtt{op} \ \mathcal{D}_2$$

where $O \in \mathcal{O}$ and `op` is an operator among `meets`, `equals`, `overlaps`, `parmin`, `parmax`, and `parmaster`. We require that each object $O \in \mathcal{O}$ appears at most once in any document specification $\mathcal{D}$.

Each operator has a dual function: First, it builds a composite document from two simpler ones. Second, it imposes constraints on the order of the starting and finishing times of the component documents. These constraints can be trivial, as in the case of the `meets` operator, or more demanding, as in the case of `equals`, where consistency has to be ensured. Before giving the translation of a document specification to a PND, let us present intuitively the meaning of the operators defined above.

- $\mathcal{D}_1$ `meets` $\mathcal{D}_2$ is the document starting when $\mathcal{D}_1$ starts, finishing when $\mathcal{D}_2$ finishes, and where the end of $\mathcal{D}_1$ coincides with the beginning of $\mathcal{D}_2$;
- $\mathcal{D}_1$ `equals` $\mathcal{D}_2$ is the document where $\mathcal{D}_1$ and $\mathcal{D}_2$ start and finish at the same time;
- $\mathcal{D}_1$ `overlaps` $\mathcal{D}_2$ is the document starting when $\mathcal{D}_1$ starts, finishing when $\mathcal{D}_2$ finishes, and where the beginning of $\mathcal{D}_2$ is strictly later than the beginning of $\mathcal{D}_1$, and the end of $\mathcal{D}_1$ is strictly later than the beginning of $\mathcal{D}_2$ and strictly earlier than the end of $\mathcal{D}_2$;
- $\mathcal{D}_1$ `parmin` $\mathcal{D}_2$ is the document where $\mathcal{D}_1$ and $\mathcal{D}_2$ start at the same time, and the one which finishes first terminates the document;
- $\mathcal{D}_1$ `parmax` $\mathcal{D}_2$ is the document where $\mathcal{D}_1$ and $\mathcal{D}_2$ start at the same time, and the one which finishes last terminates the document;
- $\mathcal{D}_1$ `parmaster` $\mathcal{D}_2$ is the document where $\mathcal{D}_1$ and $\mathcal{D}_2$ start at the same time, and the document finishes whenever $\mathcal{D}_1$ does.

Modeling With each media object $O_i, i = 1, ..., n$ we associate a clock x_i. Also, given a set of clocks X, we denote by $X := 0$ the resetting of each clock in X to zero.

We now define the translation of a document specification $\mathcal{D}$ to a PND $\mathcal{N}$. The definition is by induction on the syntax of $\mathcal{D}$. The media object O_i is translated to the net shown in figure 16.

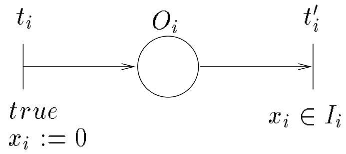


Fig. 16. The PND corresponding to the basic media object O_i.

In order to construct the PND for a document $\mathcal{D}_1$ `op` $\mathcal{D}_2$, we assume having already the PND $\mathcal{N}_1$ and $\mathcal{N}_2$ corresponding to $\mathcal{D}_1$ and $\mathcal{D}_2$, respectively. These nets have the general form shown in figure 17, that is, a single starting transition t_i, a single finishing transition t'_i guarded by g_i, and a body displayed as a dashed-line box in the figures. All the transitions are delayable, apart from the initial transition which is eager. Also, we assume that this is the case for all the PND resulting from the constructions shown in the sequel. It is easy to see that the PND of a basic object conforms to this general scheme. The constructions that are presented below preserve this general scheme.

Figures 18, 19 and 20 show the PND corresponding to $\mathcal{D}_1$ `meets` $\mathcal{D}_2$, $\mathcal{D}_1$ `equals` $\mathcal{D}_2$ and $\mathcal{D}_1$ `overlaps` $\mathcal{D}_2$, respectively. For the operators `parmin`,

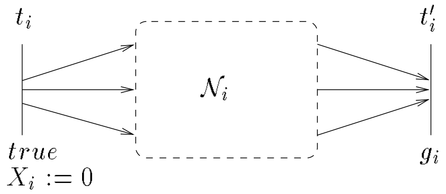


Fig. 17. The general form of PND $\mathcal{N}_i$ corresponding to a document $\mathcal{D}_i$.

`parmax` and `parmaster` the construction is identical to the one for `equals`, with the difference that the guard $g_1 \wedge g_2$ of the finishing transition t' is replaced by $\mathrm{MIN}(g_1, g_2)$, $\mathrm{MAX}(g_1, g_2)$ and $\mathrm{MASTER}(g_1, g_2)$, respectively.

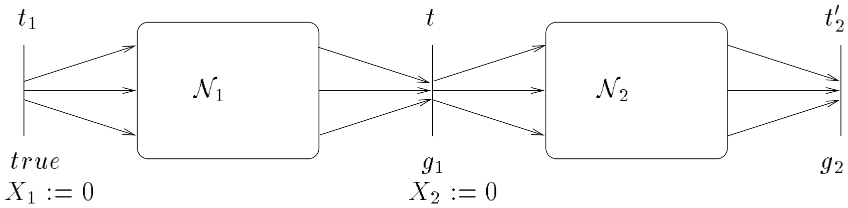


Fig. 18. The PND corresponding to document $\mathcal{D}_1$ `meets` $\mathcal{D}_2$.

Verification Given a document specification $\mathcal{D}$, we are interested in checking its *consistency*, that is, whether the temporal constraints imposed by the various operators and the duration intervals of each basic object are compatible. For instance, the specification O_1 `equals` O_2 is consistent if and only if the duration intervals I_1 and I_2 have non-empty intersection.

To check consistency, we proceed as follows. We first construct the net $\mathcal{N}$ corresponding to the specification $\mathcal{D}$. Next, we build the TAD $\mathcal{A}$ associated with $\mathcal{N}$ and add two extra locations `Begin` and `End` to $\mathcal{A}$. The former is the initial location, source of the (unique) edge of $\mathcal{A}$ corresponding to the starting transition of $\mathcal{N}$. `End` is the target location of the (unique) edge of $\mathcal{A}$ corresponding to the finishing transition of $\mathcal{N}$. `End` has no outgoing edges. Finally, we check whether `End` is reachable from `Begin`. If this is the case then $\mathcal{D}$ is consistent and we also obtain a sample execution scenario in the form of a run of the automaton $\mathcal{A}$. Otherwise, the specification is inconsistent. The reachability test is performed using the real-time verification tool Kronos.

An example

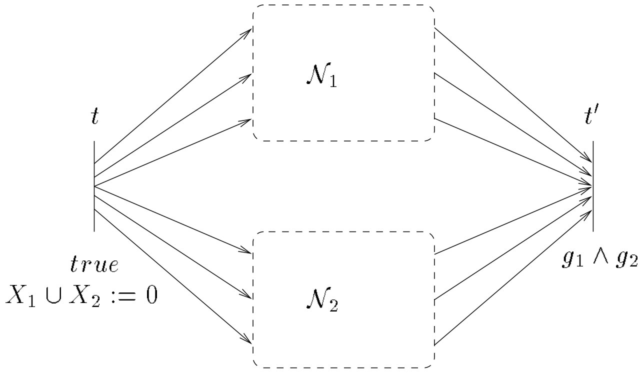


Fig. 19. The PND corresponding to document $\mathcal{D}_1$ `equals` $\mathcal{D}_2$.

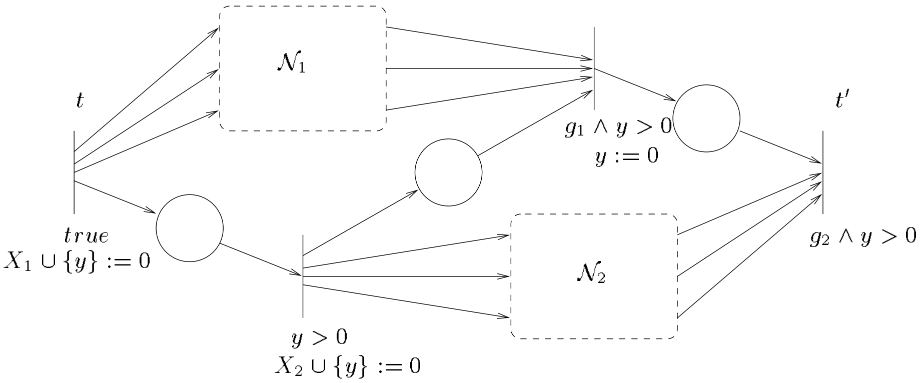


Fig. 20. The PND corresponding to document $\mathcal{D}_1$ `overlaps` $\mathcal{D}_2$.

Description. We consider the following document specification:

$$\begin{aligned}
\mathcal{D} \ &\equiv \mathcal{D}_1 \ \texttt{meets} \ \mathcal{D}_2 \\
\mathcal{D}_1 &\equiv A \ \texttt{equals} \ \big(B \ \texttt{parmax} \ (C \ \texttt{parmin} \ D)\big) \\
\mathcal{D}_2 &\equiv \big(E \ \texttt{meets} \ (F \ \texttt{equals} \ G)\big) \ \texttt{parmaster} \ (H \ \texttt{starts} \ O)
\end{aligned}$$

where:

- $\mathcal{D}$ is a document composed of two "scenes", that is, two sub-documents $\mathcal{D}_1$ and $\mathcal{D}_2$.
- $\mathcal{D}_1$ is the introduction, composed of four media objects, namely, a video clip A, a sound clip B, a piece of music C and a user button D. The intention is that the video A is played in parallel with its sound B, while at the same time music is heard in the background. The user can stop the music by pressing the button.
- $\mathcal{D}_2$ is the body of the document, composed of five media objects, namely, a still picture E followed by a video clip F and its sound clip G, which determine the presentation of an animation H and a diagram O.
- The duration intervals of the objects are as follows:

$$\begin{array}{lllll}
A : [15,17] & B : [14,16] & C : [9,11] & D : [10,13] & \\
E : [5,7] & F : [3,6] & G : [4,7] & H : [6,12] & O : [11,\infty)
\end{array}$$

- $\mathcal{D}'$ `starts` $\mathcal{D}''$ is a macro-notation for ($\mathcal{D}'$ `meets`R) `equals` $\mathcal{D}''$, where R is a "dummy" media object of null content having an arbitrary duration in $(0,\infty)$.

Modeling. The specification $\mathcal{D}$ is modeled as the PND shown in figure 21. For clarity reasons, we have reduced the number of clocks in this example to the least possible. Indeed, since A, B, C, D start simultaneously, their respective clocks have the same value, thus, they can be replaced by the same clock, say, x. Clock x is re-used for E, H, O, while clock y is associated to F, G. Finally, a clock z is associated to the dummy object used for `starts`.

All transitions are delayable and their guards are as follows:

$$\begin{aligned}
g_1 &\equiv (x \in I_A) \wedge \mathrm{MAX}\big((x \in I_B), \mathrm{MIN}\big((x \in I_C), (x \in I_D)\big)\big) \\
g_2 &\equiv (x \in I_E) \\
g_3 &\equiv (x \in I_H) \\
g_4 &\equiv \mathrm{MASTER}\big((y \in I_F \wedge y \in I_G), (z > 0 \wedge x \in I_O)\big)
\end{aligned}$$

After replacing operators MIN, MAX, MASTER by their definitions and eliminating all existential quantifiers, we obtain:

$$\begin{aligned}
g_1 &\equiv 15 \le x \le 16 \\
g_2 &\equiv 5 \le x \le 7 \\
g_3 &\equiv 6 \le x \le 12 \\
g_4 &\equiv 4 \le y \le 6
\end{aligned}$$

Therefore, we see that clock z is not needed.

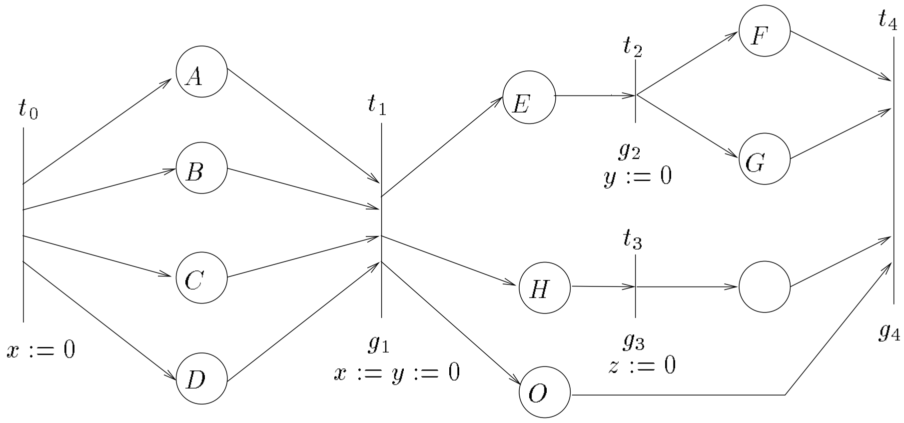


Fig. 21. Example multimedia specification translated into a PND.

Consistency analysis. In order to verify the consistency of the specification, we translate the PND of figure 21 to the TAD shown in figure 22. Then, using

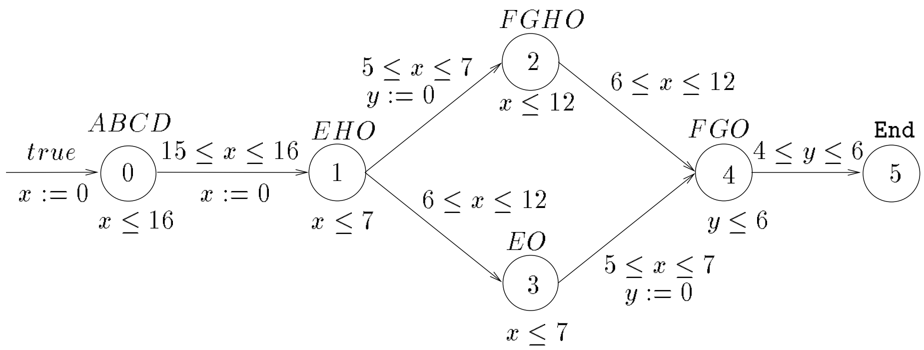


Fig. 22. The TAD corresponding to the PND of figure 21.

KRONOS, we find that the final location `End` is indeed reachable, and we are given the following sample execution scenario, in form of a *symbolic trail*. The latter is made of a sequence of *symbolic states*, that is, pairs of a control location and a clock guard. Each symbolic state is followed by its time successor, which is in turn followed by an action successor.

$\langle\ 0,\ x = 0$ `and` $y = 0\ \rangle$
$\langle\ 0,\ 15 \leq x$ `and` $x \leq 16$ `and` $x = y\ \rangle$
  $15 \leq x$ `and` $x \leq 16\ \Rightarrow$ `end_ABCD; reset`$\{x\}$`; goto` 1
$\langle\ 1,\ x = 0$ `and` $15 \leq y$ `and` $y \leq 16\ \rangle$
$\langle\ 1,\ 5 \leq x$ `and` $x \leq 7$ `and` $x + 15 \leq y$ `and` $x \leq y + 16\ \rangle$
  $5 \leq x$ `and` $x \leq 7\ \Rightarrow$ `end_E; reset`$\{y\}$`; goto` 2
$\langle\ 2,\ 5 \leq x$ `and` $x \leq 7$ `and` $y = 0\ \rangle$
$\langle\ 2,\ 6 \leq x$ `and` $x \leq 12$ `and` $y \leq 6$ `and` $x \leq y + 7$ `and` $y + 5 \leq x\ \rangle$
  $6 \leq x$ `and` $x \leq 12\ \Rightarrow$ `end_H; reset`$\{\}$`; goto` 4
$\langle\ 4,\ 6 \leq x$ `and` $x \leq 12$ `and` $y \leq 6$ `and` $x \leq y + 7$ `and` $y + 5 \leq x\ \rangle$
$\langle\ 4,\ 4 \leq y$ `and` $y \leq 6$ `and` $x \leq y + 7$ `and` $y + 5 \leq x\ \rangle$
  $4 \leq y$ `and` $y \leq 6\ \Rightarrow$ `end_FGO; reset`$\{\}$`; goto` 5
$\langle\ 5,\ 4 \leq y$ `and` $y \leq 6$ `and` $x \leq y + 7$ `and` $y + 5 \leq x\ \rangle$
$\langle\ 5,\ 4 \leq y$ `and` $x \leq y + 7$ `and` $y + 5 \leq x\ \rangle$

5 Conclusions

The paper proposes a methodological framework for modeling urgency in timed systems. Urgency is an essential feature of timed systems and is related to their capability of waiting before executing actions. Compared to untimed systems where waiting is asynchronous (indefinite waiting of a process is usually allowed), waiting times are the same in all components of a timed system. Incompatibility of time progress requirements for the processes of a system may lead to inconsistency in specifications.

The main thesis of the paper is that many different ways of composing time progress conditions are useful in practice. Furthermore, time progress condition description should not be dissociated from action description. This leads to the definition of TAD which are timed automata composed of timed transitions, transitions specified in terms of two related conditions expressing respectively, possibility and forcing of execution by stopping time progress. The TAD are a subclass of timed automata that satisfy the time reactivity condition meaning that from any state as long as there are no actions enabled, time can progress.

The proposed methodology is based on the idea that complex timed systems can be obtained as the composition of elementary ones (timed transitions) by means of choice and synchronization operations. The latter allow to define the guard and the deadline of a synchronization action in terms of the guards and deadlines of the synchronizing actions. Apart from AND-synchronization that corresponds to the commonly used conjunctive synchronization, other synchronization modes are shown to be of practical interest as they have been introduced in timed models such as the timed extensions of Petri nets. These synchronization modes can be expressed in terms of AND-synchronization if auxiliary states (and transitions) are added to represent information encoded by modalities in the expression of synchronization guards. However, this may lead to complex constructions and make specifications less legible. Thus, the different synchronization modes are at least an interesting macro-notation, especially for systems with loosely coupled components where coordination is realized by mechanisms seeking consensus and flexibility e.g., protocols. In fact, the modal formulas in synchronization guards can be considered as the abstract specifications of a protocol used to implement the described coordination.

The paper contributes to clarifying the notion of urgency and proposes the mechanisms that are necessary for a "natural" specification of timed systems. It shows amongst others, that for general timed systems specification a rich methodological framework is necessary that includes new concepts and constructs that are not applicable to untimed systems. In fact, compositional description of untimed specifications can be extended in many different manners to timed specifications, as shown by several examples. It is remarkable that the composition mechanisms defined initially for timed automata or process algebras are obtained by lifting directly the corresponding mechanisms for untimed systems (conjunction of guards and time progress conditions for synchronization) This contrasts with ad hoc flexible synchronization mechanisms added to Petri nets or to logical specification languages. We believe that our results allow to compare and better understand the relations between the existing timed formalisms and can be a basis of a framework for compositional specification of timed systems.

References

[ABRW91] N.C. Audsley, A. Burns, M.F. Richardson, and A.J. Wellings. Deadline-monotonic scheduling. In *Proc. 8th IEEE Workshop on Real-time Operating Systems and Software*, 1991.

[ACD93] R. Alur, C. Courcoubetis, and D.L. Dill. Model checking in dense real time. *Information and Computation*, 104(1):2–34, 1993.

[All83] J.F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, November 1983.

[BS97a] S. Bornot and J. Sifakis. On the composition of hybrid systems (complete version). In *International NATO Summer School on "Verification of Digital and Hybrid Systems", Antalya, Turkey*, 1997.

[BS97b] S. Bornot and J. Sifakis. Relating time progress and deadlines in hybrid systems. In *International Workshop, HART'97*, pages 286–300, Grenoble, France, March 1997. Lecture Notes in Computer Science 1201, Spinger-Verlag.

[HNSY94] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.

[JLSIR97] M. Jourdan, N. Layaïda, L. Sabry-Ismail, and C. Roisin. Authoring and presentation environment for interactive multimedia documents. In *Proc. of the 4th Conf. on Multimedia Modelling*, Singapore, November 1997. World Scientific Publishing.

[Mer74] P. Merlin. A study of the recoverability of computer systems. Master's thesis, University of California, Irvine, 1974.

[SDdSS94] P. Sénac, M. Diaz, and P. de Saqui-Sannes. Toward a formal specification of multimedia scenarios. *Annals of telecomunications*, 49(5-6):297–314, 1994.

[Sif77] J. Sifakis. Use of petri nets for performance evaluation. In H. Beilner and E. Gelenebe, editors, *Measuring, modelling and evaluating computer systems*, pages 75–93. North-Holland, 1977.

[SY96] J. Sifakis and S. Yovine. Compositional specification of timed systems. In *13th Annual Symposium on Theoretical Aspects of Computer Science, STACS'96*, pages 347–359, Grenoble, France, February 1996. Lecture Notes in Computer Science 1046, Spinger-Verlag.