Real-Time Systems, 23, 55–84, 2002
© 2002 Kluwer Academic Publishers. Manufactured in The Netherlands.

# Scheduler Modeling Based on the Controller Synthesis Paradigm

K. ALTISEN altisen@imag.fr
*VERIMAG, 2 av. de Vignate, 38610 Gières, France*

G. GÖSSLER goessler@imag.fr
*VERIMAG, 2 av. de Vignate, 38610 Gières, France*

J. SIFAKIS sifakis@imag.fr
*VERIMAG, 2 av. de Vignate, 38610 Gières, France*

**Abstract.** The controller synthesis paradigm provides a general framework for scheduling real-time applications. Schedulers can be considered as controllers of the applications; they restrict their behavior so that given scheduling requirements are met.

We study a modeling methodology based on the controller synthesis paradigm. The methodology allows to get a correctly scheduled system from timed models of its processes in an incremental manner, by application of composability results which simplify schedulability analysis. It consists in restricting successively the system to be scheduled by application of constraints defined from scheduling requirements. The latter are a conjunction of schedulability requirements that express timing properties of the processes and policy requirements about resource management.

The presented methodology allows a unified view of scheduling theory and approaches based on timing analysis of models of real-time applications.

**Keywords:** modeling real-time systems, scheduler design, controller synthesis, dynamic priorities, composability

## 1. Introduction

Schedulers coordinate the execution of system activities, so that requirements about their temporal behavior are met. Guaranteeing their correctness is essential for the development of dependable real-time systems. Well established theory and scheduling algorithms have been successfully applied to real-time systems development. Existing scheduling theory requires the application to fit into the mathematical framework of the schedulability criterion (e.g., all processes are supposed periodic, worst case execution times are known). Studies to relax such hypotheses have been carried out, but no unified approach has been proposed so far. To overcome these limitations, an alternative approach consists in extracting a scheduler from an abstract timed model of a real-time application by using analysis or synthesis tools (Ben-Abdalla et al., 1999; Bertin et al., 2000; Henzinger et al., 2001; Jensen et al., 2000; Kwak et al., 1998; Niebert and Yovine, 2000).

The controller synthesis paradigm (Ramadge and Wonham, 1987) provides a general framework for scheduling. A scheduler can be considered as a controller of the real-time application which restricts its behavior so that given *scheduling requirements* are met. Behavior restriction essentially amounts to resolving non-determinism due to concurrent access of processes to shared resources.

To apply the controller synthesis paradigm, it is necessary to use a timed model representing the dynamic behavior of the real-time application. The scheduler for a given set of scheduling requirements is also a timed system which observes the state of the application and adequately restricts its behavior by triggering *controllable* actions that is, actions giving access to shared resources (see Figure 1). The role of the scheduler consists precisely in observing the application and maintaining the requirements satisfied in spite of ''disturbances'' of the environment and of internal actions of the application, usually represented by *uncontrollable* actions, such as process arrival or process termination.

We have shown in (Altisen et al., 1999) how schedulers can be computed by application of a synthesis method to systems represented by well-timed models that is, to timed models where time can always progress. For such systems, scheduling requirements can be characterized as a safety property expressing the fact that a constraint (state predicate) $K$ always holds. The main result is that there exists a scheduler maintaining $K$ if there exists a non-empty *control invariant* $K'$ which implies $K$. The control invariant $K'$ represents the set of the states from which $K'$ (and thus $K$) can be preserved, in the sense that if the application is initially at a state satisfying $K'$ then it is possible to remain in states of $K'$ by triggering controllable actions preserving $K'$, and it is not possible to violate $K'$ by uncontrollable actions.

Control invariants implying a given scheduling constraint $K$ can be computed by controller synthesis methods (Altisen et al., 1999, 2000; Lin and Wonham, 1988; Maler et al., 1995; Ramadge and Wonham, 1987). The existence of a non-empty control invariant is a necessary and sufficient condition for the existence of a scheduler. The latter can be constructed from the control invariant and the timed model of the application. A common limitation of controller synthesis algorithms is their complexity that makes problematic their application to large systems.

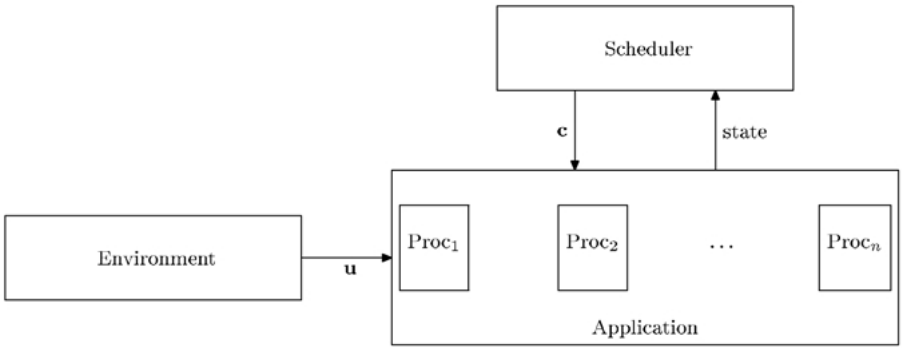


*Figure 1.* Interactions between the environment, the application and its scheduler.

We use the controller synthesis paradigm as a unifying framework for scheduling real-time applications. According to this paradigm, a scheduler can be specified as a pair consisting of the timed model of the application to be scheduled and of a constraint $K$ characterizing scheduling requirements. We study a modeling methodology which, from such an initial specification allows finding a scheduler by circumventing as much as possible complexity problems. The paper contributes along the following three directions.

First, it provides a notation and a methodology for modeling the application as a timed system composed of the processes to be scheduled, their resources and the associated synchronization constraints. Timing constraints relate in particular process execution speed with the dynamics of their external environment. For the sake of simplicity, we use discrete time models. The methodology can be adapted to continuous time models modulo some additional problems related to well-timedness of descriptions.

The proposed notation uses results presented in Altisen et al. (2000) and allows an incremental description of the application, starting from its processes and then adding timing constraints and synchronization constraints associated to the resources. It allows modeling in a direct manner dynamic priorities and preemption as well as concepts such as urgency, idling and timeliness. Furthermore, the models are well-timed, by construction.

Second, the paper shows that scheduling requirements can be characterized as the invariance of a constraint which is the conjunction of two classes of constraints: schedulability requirements $K_{\text{sched}}$ and a (possibly empty) set of constraints characterizing a particular scheduling policy $K_{\text{pol}}$. $K_{\text{sched}}$ characterizes the dynamic properties of the application to be satisfied by the scheduler, relating execution times, process arrival times, and deadlines. $K_{\text{pol}}$ deals with the management of shared resources and can be decomposed into the conjunction of two (possibly empty) classes of constraints: conflict resolution constraints $K_{\text{res}}$ that determine the rules for granting a resource to conflicting processes, and admission control constraints $K_{\text{adm}}$ that determine when the scheduler considers a non-conflicting request for a free resource.

Finally, the paper provides conditions under which an incremental modeling methodology can be applied to get the scheduler. The search for control invariants implying given scheduling requirements of the form $K_{\text{sched}} \wedge K_{\text{pol}}$ is decomposed into two steps. A first step for computing a scheduler maintains the scheduling policy specified by $K_{\text{pol}}$. This step does not require the application of synthesis algorithms, as $K_{\text{pol}}$ is shown to be a control invariant. The second step aims at establishing that the system scheduled according to $K_{\text{pol}}$ meets the schedulability requirements $K_{\text{sched}}$. This step requires in general, the application of synthesis or verification techniques that can be carried out by existing timing analysis tools such as KRONOS (Daws et al., 1996), UPPAAL (Jensen et al., 2000), HYTECH (Henzinger et al., 1997).

The paper is organized as follows. Section 2 presents models and basic results about controller synthesis that are used throughout the paper. The main results concern control invariants and their composability properties which play an instrumental role in the modeling methodology. Section 3 focuses on modeling issues of the real-time application to be scheduled including modeling of the processes, the associated timing constraints, and the resource management and synchronization. Section 4

presents a method for specifying scheduling policies and computing the associated scheduler. The application of the method is illustrated with several examples. Section 5 proposes a method for specifying schedulability requirements and getting a correct scheduler by using synthesis or verification tools. Section 6 illustrates the method on an example.

## 2. Controller Synthesis

### *2.1. Timed System*

To model scheduling algorithms, we use reactive timed systems with two kinds of actions as in Altisen et al. (1999): controllable actions that can be triggered by the scheduler, and uncontrollable actions that are internal actions of the processes to be scheduled or actions of the environment. Controllable actions are typically resource allocations while uncontrollable actions are process arrival and termination.

Both controllable and uncontrollable actions are subject to timing constraints expressed in terms of natural variables called *timers*. The rates of timers may take the values 0 or 1, as specified by a Boolean vector.

**Definition 2.1** *(X-constraint).* Let $X$ be a finite set of timers, $\{x_1, \ldots, x_m\}$, natural variables defined on the set of naturals $\mathbb{N} = \{0, 1, 2, \ldots\}$. An *X-constraint* is a predicate $C$ generated by the grammar $C ::= x \leq d \mid x - y \leq d \mid C \wedge C \mid \neg C$, where $x, y \in X$, $d$ is an integer.

**Definition 2.2** *(Timed system). A timed system* consists of the following:

1. An untimed labeled transition system $(S, A, T)$ where $S$ is a finite set of control states, $A$ is a finite vocabulary of actions partitioned into two sets of controllable and uncontrollable actions noted $A^c$ and $A^u$, and $T \subseteq S \times A \times S$ is an untimed transition relation.

2. A finite set of timers $X = \{x_1, \ldots, x_m\}$, as in Definition 2.1.

3. A function $\mathbf{b}$ mapping $S$ into $\{0, 1\}^m$. The image of $s \in S$ by $\mathbf{b}$ denoted $\mathbf{b}_s$ is a Boolean rate vector.

4. A labeling function $h$ mapping untimed transitions of $T$ into timed transitions: $h(s, a, s') = (s, a, g, \tau, r, s')$, where $g$ is an $X$-constraint called *guard*; $r \subseteq X$ is a set of timers to be *reset*; $\tau \in \{\delta, \varepsilon\}$ is an *urgency type*, respectively delayable and eager.

*Semantics.* A timed system defines a *transition graph* $(\mathscr{V}, \mathscr{E})$ constructed as follows. $\mathscr{V} = S \times \mathbb{N}^m$, that is, vertices $(s, \mathbf{x})$ are states of the timed system.

The set $\mathscr{E} \subseteq \mathscr{V} \times (A \cup (\mathbb{N} \setminus \{0\})) \times \mathscr{V}$ of the edges of the graph is partitioned into three classes of edges: $\mathscr{E}^c$ controllable, $\mathscr{E}^u$ uncontrollable, and $\mathscr{E}^t$ timed, corresponding respectively to the case where the label is a controllable action, an uncontrollable action, and a strictly positive integer.

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 untimed transitions departing from $s$. Also, let $h(s, a_j, s_j) = (s, a_j, g_j, \tau_j, r_j, s_j)$. For all $j \in J$, $((s, \mathbf{x}), a_j, (s_j, \mathbf{x}[r_j])) \in \mathscr{E}^c \cup \mathscr{E}^u$ iff $g_j(\mathbf{x})$ holds and $\mathbf{x}[r_j]$ is the timer valuation obtained from $\mathbf{x}$ when all the timers in $r_j$ are set to zero and the others are left unchanged.

To define $\mathscr{E}^t$, we use the predicate $\varphi$, called *time progress function*. The notation $\varphi((s, \mathbf{x}), t)$ means that time can progress from state $(s, \mathbf{x})$ by $t$.

$$\varphi((s,\mathbf{x}),t) \Longleftrightarrow$$
$$\bigwedge_{j \in J} \begin{cases} \tau_j = \delta \Rightarrow \forall t' \in \{0, \ldots, t-1\} \cdot & \neg g_j(\mathbf{x} + t'\mathbf{b}_s) \vee \\ & g_j(\mathbf{x} + (t'+1)\mathbf{b}_s) \\ \tau_j = \varepsilon \Rightarrow \forall t' \in \{0, \ldots, t-1\} \cdot & \neg g_j(\mathbf{x} + t'\mathbf{b}_s) \end{cases}$$

where $\mathbf{x} + t\mathbf{b}_s$ is the valuation obtained from $\mathbf{x}$ by increasing by $t$ the timer values for which $\mathbf{b}_s$ elements are equal to one. We define $\mathscr{E}^t$ such that $((s, \mathbf{x}), t, (s, \mathbf{x} + t\mathbf{b}_s)) \in \mathscr{E}^t$ if and only if $\varphi((s, \mathbf{x}), t)$. The above definition means that at control state $s$, time cannot progress beyond the falling edge of a delayable guard, or whenever an eager guard is enabled.

Timed systems are automata extended with time variables as timed or hybrid automata (Alur et al., 1995; Alur and Dill, 1994) where time variables are real-valued. We prefer using discrete time variables for the sake of simplicity. The presented approach is also applicable to dense time models modulo some technical problems related to time density. Another difference between our model and dense timed and hybrid automata is well-timedness, that is, time can always progress at a state where no transition is enabled. This property is crucial for the expression of schedulability requirements as a safety property (see Section 5).

We will usually denote by TS a timed system. $\mathrm{TS}^c$ (resp. $\mathrm{TS}^u$) represents the timed system consisting of the controllable (resp. uncontrollable) transitions of TS only.

**Lemma 2.1** *If $\varphi$, $\varphi^{\mathrm{c}}$, and $\varphi^{\mathrm{u}}$ are respectively, the time progress functions of TS, $\mathrm{TS}^c$, and $\mathrm{TS}^u$, then $\varphi = \varphi^{\mathrm{c}} \wedge \varphi^{\mathrm{u}}$.*

**Example 2.1** *(A basic process).* Figure 2 represents as a timed system a periodic process $P$ of period $T$, execution time $E$, and deadline of $D (0 < E \leq D \leq T)$.

The timed system has three control states, $s$, $w$, and $u$ where $P$ is respectively, sleeping, waiting and executing. The actions $a$, $b$, and $e$ stand for arrive, begin, and end. The timer $x$ is used to measure execution time while the timer $t$ measures the time elapsed since process arrival. In all states, both timers progress. The only controllable action is $b$.

By convention, transition labels are of the form $a^{\mathbf{y}}$, $g^{\tau}$, $r$, where $\mathbf{y}$ can be $\mathbf{u}$ (uncontrollable) or $\mathbf{c}$ (controllable), $\tau$ is an urgency type, and $r$ is a set of timers to be reset. The upperscript $\mathbf{c}$ may be omitted, as well as the set $r$ if it is empty.

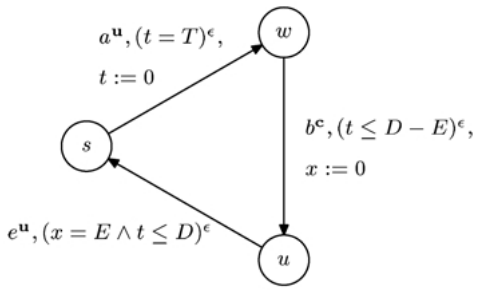


*Figure 2*. A periodic process.

Notice that transition $b$ is eager, which means that the process executes as soon as possible. If $b$ is chosen delayable, waiting at $w$ is possible for time $D - E$.

**Definition 2.3** *(Composition of timed systems).* Let $\mathrm{TS}_i = (S_i, A_i, T_i, X_i, \mathbf{b}_i, h_i)$ for $i \in \mathscr{I}$, where $\mathscr{I} = \{1, \ldots, n\}$ is a finite index set, be a set of timed systems and let $\Sigma$ be a set of subsets of $\cup_{i \in \mathscr{I}} A_i$ such that any $\mathbf{a} \in \Sigma$ is not empty and contains at most one action of each process. $\Sigma$ is a set of *synchronization actions*. We say that $TS = (S, A, T, X, \mathbf{b}, h)$ is the composition of the timed systems $\mathrm{TS}_i$, $i \in \mathscr{I}$ with respect to $\Sigma$ if

$S = S_1 \times \cdots \times S_n$; $A = A_1 \cup \cdots \cup A_n \cup \Sigma$; $X = X_1 \cup \cdots \cup X_n$;
For all $s = (s_1 \ldots s_n) \in S$ and $x \in X$,
  $\mathbf{b}_s[x] = 0$ if $\exists i \in \mathscr{I} \cdot \mathbf{b}_{i,s_i}[x] = 0$; $\mathbf{b}_s[x] = 1$ otherwise;
For all $s = (s_1 \ldots s_n) \in S, s' = (s'_1 \ldots s'_n) \in S$,
(interleaving transitions) $\forall i \in \mathscr{I} \forall a \in A_i \cdot t = (s, a, s') \in T \Longleftrightarrow$
  $t_i = (s_i, a, s'_i) \in T_i \wedge \forall j \in \mathscr{I} \setminus \{i\} \cdot s'_j = s_j \wedge$
  $\forall \mathbf{a} \in \Sigma \cdot (a \in \mathbf{a} \Rightarrow \exists k \in \mathscr{I}\ \exists a_k \in \mathbf{a} \cap A_k\ \forall s''_k \in S_k \cdot (s_k, a_k, s''_k) \notin T_k)$;
  if $h_i(t_i) = (s_i, a, g, \tau, r, s'_i)$, then $h(t) = (s, a, g, \tau, r, s')$;
(synchronization transitions) $\forall \mathbf{a} \in \Sigma \cdot t = (s, \mathbf{a}, s') \in T \Longleftrightarrow$
  $\forall i \in \mathscr{I} \cdot (t_i = (s_i, a_i, s'_i) \in T_i \wedge a_i \in \mathbf{a} \vee s_i = s'_i \wedge \mathbf{a} \cap A_i = \emptyset) \wedge$
  $\forall \mathbf{a}' \in \Sigma \cdot (\mathbf{a} \subsetneq \mathbf{a}' \Rightarrow \exists k \in \mathscr{I}\ \exists a_k \in \mathbf{a}' \cap A_k\ \forall s''_k \in S_k \cdot (s_k, a_k, s''_k) \notin T_k)$;
  if for any synchronizing transition $t_i$, $h_i(t_i) = (s_i, a_i, g_i, \tau_i, r_i, s'_i)$,
  then $h(t) = (s, \mathbf{a}, \bigwedge_{a_i \in \mathbf{a}} g_i, \max_{a_i \in \mathbf{a}}\{\tau_i\}, \cup_{a_i \in \mathbf{a}} r_i, s')$,
  where $\max\{\varepsilon, \tau\} = \varepsilon$.

Non-synchronizing actions, $a \in A \setminus \Sigma$, keep their controllability type, whereas synchronization actions $\mathbf{a} \in \Sigma$ are controllable if and only if all elements of $\mathbf{a}$ are controllable.

We write $\mathrm{TS} = \|_{\Sigma} \{\mathrm{TS}_i\}_{i \in \mathscr{I}}$ to denote the timed system of processes defined above, and $a_j \mid \cdots \mid a_k$ for the synchronization action $\{a_j, \ldots, a_k\} \in \Sigma$.

Intuitively, interleaving is possible only if synchronization is not possible. Synchronization transitions $t = (s, \mathbf{a}, s')$ must satisfy two conditions. First, all processes participating in action $\mathbf{a}$ can perform a transition labeled by some element of $\mathbf{a}$, and all

other processes remain idle. Second, **a** must be maximal among the actions in $\Sigma$ that are possible from $s$ (maximal progress).

### 2.2. *Control Invariants*

In this paragraph, we introduce and study concepts used to define the synthesis problem. Restriction of a timed system by a constraint $K$ is used to model the effect of a scheduler on the real-time application. To maintain the constraint $K$, the scheduler triggers a controllable action only if its execution leads to states satisfying $K$.

**Definition 2.4** *(Constraint).* Given a timed system with a set of timers $X$, a set of control states $S$, and a set of transitions $T$, a *constraint* is a state predicate represented as an expression of the form

$$K = \bigvee_{s \in S} s \wedge K^s \wedge \bigwedge_{(s,a,s') \in T} [a](K^{sas'})$$

$K^s$ and $K^{sas'}$ are $X$-constraints, $s$ is (also) the Boolean denoting presence at control state $s$, and $[a](K^{sas'})$ is the predicate defined by, for some state $(s, \mathbf{x})$,

$$[a]\left(K^{sas'}\right)(s,\mathbf{x}) \Longleftrightarrow \left(\exists((s,\mathbf{x}),a,(s',\mathbf{x}')) \in \mathscr{E} \Rightarrow K^{sas'}(\mathbf{x})\right)$$

That is, the guards of all transitions of the form $(s, a, s')$ imply $K^{sas'}$.

A constraint $K$ is called a *transition constraint* if for all control states $s \in \mathrm{S}, K^s = \text{true}$. It is called a *state constraint* if for all transitions $(s, a, s') \in T, K^{sas'} = \text{true}$.

We note $K(s)$ for some $s \in S$, the $X$-constraint defined by the projection of $K$ on $s$: $\forall \mathbf{x} \in \mathbb{N}^m, K(s)(\mathbf{x}) \Longleftrightarrow K(s,\mathbf{x})$.

**Definition 2.5** *(Restriction).* Let TS be a timed system and $K$ be a constraint. The *restriction* of TS by $K$ denoted TS/$K$, is the timed system TS where the guard $g$ of any controllable transition $(s, a, g, \tau, r, s')$, is replaced by $g'$ where

$$g'(\mathbf{x}) = g(\mathbf{x}) \wedge K^{sas'}(\mathbf{x}) \wedge K^{s'}(\mathbf{x}[r])$$

Notice that in the restriction TS/$K$, the states right after, and right before the execution of a controllable transition satisfy $K$. Moreover, it follows from the definition that $(\mathrm{TS}/K_1)/K_2 = \mathrm{TS}/(K_1 \wedge K_2)$. The symbol ‘‘$=$’’ between timed systems always means syntactical equality modulo guard equivalence.

The two following notions of invariant are useful for the definition of the synthesis problem. For a given timed system, a proper invariant is a constraint preserved by all its transitions while a control invariant is a constraint that cannot be violated by uncontrollable transitions and time progress if controllable transitions are triggered only when they lead to states satisfying $K$. For systems without controllable actions the

notions of proper and control invariant coincide and correspond to the usual notion of invariant.

**Definition 2.6** *(Proper invariant).* Let TS be a timed system and $K$ be a constraint. $K$ is a *proper invariant* of TS, denoted by TS $\models$ inv($K$), if $K$ is preserved by the edges of $\mathscr{E}$, the set of edges of the transition graph of TS, i.e.,

$$\forall(s,\mathbf{x}) \cdot \left(\mathrm{K}^s(\mathbf{x}) \wedge \exists((s,\mathbf{x}),\gamma,(s',\mathbf{x}')) \in \mathscr{E}\right) \Rightarrow \\ \mathrm{K}^{s'}(\mathbf{x}') \wedge \left(\gamma \in A \Rightarrow \mathrm{K}^{s\gamma s'}(\mathbf{x})\right)$$

**Definition 2.7** *(Control invariant).* Let TS be a timed system and $K$ be a constraint. $K$ is a *control invariant* of TS if TS$/K \models$ inv($K$).

**Lemma 2.2** *If $K$ is a proper invariant of a timed system TS, then $K$ is a control invariant of TS.*

Notice that control invariants are not proper invariants, in general.

**Proposition 2.1** For any timed system TS and constraint $K$ such that TS$^u \models$ inv(K), $K$ is a control invariant of TS (i.e., TS$/K \models$ inv($K$)).

**Proof:** *(sketch)*: Assume that $K^s(\mathbf{x})$ holds for some state $(s,\mathbf{x})$. To prove TS$/K \models$ inv($K$) it must be shown that $K$ is preserved in TS/$K$ by (1) controllable, (2) uncontrollable, and (3) timed edges of TS/$K$. By construction of TS/$K$, (1) is true. From TS$^u \models$ inv($K$), (2) and (3) follow. ■

*The synthesis problem.* According to results in Maler et al. (1995), finding a controller which maintains a constraint $K$ for a timed system TS amounts to finding a non-empty control invariant $K'$ of TS which implies $K$, i.e., $K' \Rightarrow K$ and TS$/K' \models$ inv($K'$).

The control invariant $K'$ represents the set of the states of TS from which it is possible to maintain $K'$ (and thus $K$) by restricting the controllable actions of TS. The system TS/$K'$ represents the controlled system. As shown in Maler et al. (1995), the controller is a function associating with states of TS the set of allowed controllable actions. This function can be specified by the set of the guards of the controllable transitions of TS/$K'$. For a given state of TS, the controller evaluates the guards to decide which controllable actions are enabled.

The results of this paragraph are illustrated in the following example.

**Example 2.2** Consider the timed system $\mathrm{TS}_{12} = \|_{\emptyset}\ \{\mathrm{TS}_1, \mathrm{TS}_2\}$ where the $\mathrm{TS}_i$ are instances of the process of Figure 2, for which the parameters $(E,T,D)$ are equal to (5, 15, 15) and (2, 5, 5), respectively.

The constraint

$$K_{\mathrm{dlf}} = [(s_1 \wedge t_1 \leq 15) \vee (u_1 \wedge x_1 \leq 5 \wedge t_1 \leq 15) \vee (w_1 \wedge t_1 \leq 10)] \\ \wedge [(s_2 \wedge t_2 \leq 5) \vee (u_2 \wedge x_2 \leq 2 \wedge t_2 \leq 5) \vee (w_2 \wedge t_2 \leq 3)]$$

expresses the fact that each one of the processes is deadlock-free: from a control state, time can progress to enable the guard of some exiting transition. It is easy to check that $K_{\text{dlf}}$ is a proper invariant of $\text{TS}_{12}$.

The constraint

$$K_{\text{mutex}} = \neg(u_1 \wedge u_2)$$

expresses mutual exclusion. Clearly it is not a proper invariant of $\text{TS}_{12}$. In fact, it can be violated from state $(w_1 u_2)$ which satisfies it. However, $K_{\text{mutex}}$ is a control invariant since only controllable transitions can violate it (i.e., $\text{TS}_{12}^{u} \models \text{inv}(K_{\text{mutex}})$). Thus, $K_{\text{mutex}}$ is a proper invariant of $\text{TS}_{12}/K_{\text{mutex}}$.

For the system $\text{TS}_{12}/K_{\text{mutex}}$, the constraint $K_{\text{dlf}}$ is not a proper invariant any more: at control state $(w_1 w_2)$ for all timer values equal to zero it may happen that process 1 is served before process 2; this implies that process 2 misses its deadline. Furthermore, $K_{\text{dlf}}$ is not even a control invariant of $\text{TS}_{12}/K_{\text{mutex}}$ since it can be checked that $(\text{TS}_{12}/K_{\text{mutex}})/K_{\text{dlf}} = \text{TS}_{12}/K_{\text{mutex}}$, which means that if $K_{\text{dlf}}$ is a control invariant of $\text{TS}_{12}/K_{\text{mutex}}$, then it is a proper invariant of $\text{TS}_{12}/K_{\text{mutex}}$. The latter is not possible by the given counter-example. A non empty control invariant $K$ such that $K \Rightarrow K_{\text{dlf}}$ can be computed by application of the synthesis algorithm of KRONOS (Altisen et al., 1999; Daws et al., 1996):

$$K = \left[\begin{array}{l} (s_1 \wedge s_2 \wedge t_1 \leq 15 \wedge t_2 \leq 5) \\ \vee\, (w_1 \wedge s_2 \wedge (t_2 \leq 3 \wedge t_1 \leq 10 \vee t_2 \leq 5 \wedge t_1 \leq t_2 + 3)) \\ \vee\, (s_1 \wedge w_2 \wedge t_1 \leq 15 \wedge t_2 \leq 3) \\ \vee\, (u_1 \wedge s_2 \wedge t_2 \leq 5 \wedge x_1 \leq 5 \wedge t_1 \leq x_1 + 10 \wedge t_2 \leq x_1 + 3) \\ \vee\, (w_1 \wedge w_2 \wedge (t_1 \leq 8 \wedge t_2 \leq 1 \vee t_2 \leq 3 \wedge t_1 \leq t_2 + 3)) \\ \vee\, (s_1 \wedge u_2 \wedge t_1 \leq 15 \wedge x_2 \leq 2 \wedge t_2 \leq x_2 + 3) \\ \vee\, (u_1 \wedge w_2 \wedge x_1 \leq 5 \wedge t_1 \leq x_1 + 10 \wedge t_2 + 2 \leq x_1) \\ \vee\, (w_1 \wedge u_2 \wedge (x_2 \leq 2 \wedge t_1 \leq x_2 + 8 \wedge t_2 \leq x_2 + 1 \vee \\ \quad x_2 \leq 2 \wedge t_1 \leq t_2 + 3 \wedge t_2 \leq x_2 + 3)) \end{array}\right.$$

The corresponding controller is specified by the controllable guards of $\text{TS}_{12}/(K_{\text{mutex}} \wedge K)$, as shown in Table 1. The guards of actions $b_1$ and $b_2$ are obtained by applying definition 2.5 of restriction.

*Table 1.* Specification of the controller maintaining deadlock-freedom in $\text{TS}_{12}/K_{\text{mutex}}$.

| Action | Guard |
|---|---|
| $b_1$ | $s_2 \wedge t_1 \leq 10 \wedge t_2 \leq 3$ |
| $b_2$ | $s_1 \wedge t_1 \leq 15 \wedge t_2 \leq 3 \vee$ |
| | $w_1 \wedge (t_1 \leq 8 \wedge t_2 \leq 1 \vee t_1 \leq t_2 + 3 \wedge t_2 \leq 3)$ |

### *2.3. Composable Invariants*

A well-known fact about proper invariants is that the conjunction of two proper invariants is a proper invariant, that is TS $\models$ inv($K_i$) for $i = 1, 2$ implies TS $\models$ inv($K_1 \wedge K_2$). This property provides a basis for compositional reasoning. However, control invariance is not preserved by conjunction. This fact is illustrated in the previous example (2.2) where both $K_{\text{dlf}}$ and $K_{\text{mutex}}$ are control invariants of $\text{TS}_{12}$. $K_{\text{dlf}}$ is not a control invariant of $\text{TS}_{12}/K_{\text{mutex}}$, and thus, $K_{\text{dlf}} \wedge K_{\text{mutex}}$ is not a control invariant of $\text{TS}_{12}$. We define composable invariants, a class of control invariants preserved by conjunction.

**Definition 2.8** *(Composable invariant).* Let TS be a timed system and $K_1$ be a constraint. $K_1$ is a *composable invariant* of TS if for all constraints $K_2$, $K_1$ is a control invariant of $\text{TS}/K_2$ (i.e., if $\text{TS}/(K_1 \wedge K_2) \models \text{inv}(K_1)$ ).

**Proposition 2.2** Let TS be a timed system and $K$ be a constraint on TS. $K$ is a composable invariant of TS iff $\text{TS}^u \models \text{inv}(K)$.

**Proof:** Let $K_1$ be a composable invariant of TS. By applying Definition 2.8 with $K_2 = \text{false}$, we obtain $\text{TS}/\text{false} = \text{TS}^u \models \text{inv}(K_1)$.

Conversely, assume that $\text{TS}^u \models \text{inv}(\text{K}_1)$ and let $\text{K}_2$ be some constraint. We show that $\text{TS}/(K_1 \wedge K_2) \models \text{inv}(K_1)$. Let $(s, \mathbf{x})$ be a state of TS such that $K_1^s(\mathbf{x})$. (1) If there exists a controllable edge $((s, \mathbf{x}),\ a_c, (s', \mathbf{x}'))$ in the transition graph of $\text{TS}(K_1 2)$, then by Definition 2.5 of restriction, $K_1^{s'}(\mathbf{x}')$ and $K_1^{sa_c s'}(\mathbf{x})$ hold. (2) An uncontrollable edge $((s, \mathbf{x}),\ a_u, (s', \mathbf{x}'))$ of $\text{TS}/(K_1 2)$ is also an uncontrollable edge of $\text{TS}^u$, thus $K_1^{s'}(\mathbf{x}')$ and $K_1^{sa_u s'}(\mathbf{x})$ hold. (3) Let $\varphi_{(\text{K}_1 \wedge \text{K}_2)}$ be the time progress function of $\text{TS}/(K_1 \wedge K_2)$. According to the Proposition 2.1, we have

$$\varphi_{(K_1 \wedge K_2)} = \varphi^c_{(K_1 \wedge K_2)} \wedge \varphi^u_{(K_1 \wedge K_2)} = \varphi^c_{(K_1 \wedge K_2)} \wedge \varphi^u$$

Thus, if $((s, \mathbf{x}), t, (s, \mathbf{x} + t\mathbf{b}_s))$ is a timed edge of $\text{TS}/(K_1 \wedge K_2)$, then it is also a timed edge of $\text{TS}^u$. Hence, $K_1^s(\mathbf{x} + t\mathbf{b}_s)$ from $\text{TS}^u \models \text{inv}(K_1)$. ■

**Corollary 2.1** For a timed system TS and constraints $K_1$ and $K_2$, $\text{TS}^u \models \text{inv}(K_1)$ and $(\text{TS}/K_1)/K_2 \models \text{inv}(K_2)$ implies that $\text{TS}/(K_1 \wedge K_2) \models \text{inv}(K_1 \wedge K_2)$. That is, if $K_1$ is composable and if $K_2$ is a control invariant of $\text{TS}/K_1$ then $(K_1 \wedge K_2)$ is control invariant of TS.

This corollary provides a basis for incremental restriction of a timed systems. To impose a control invariant $K_1 \wedge K_2$ on TS, if $K_1$ is a composable invariant of TS, the restriction by a control invariant $K_2$ does not destroy the invariance of $K_1$.

**Proposition 2.4** Let TS be a timed system and K be a transition constraint. If for all uncontrollable transitions $(s, a, g, \tau, r, s') \in T, a \in A^u, g \Rightarrow K^{sas'}$, then $K$ is a composable control invariant of TS.

**Proof:** Composability obviously comes from $\mathrm{TS}^u \models \mathrm{inv}(K)$ since for all control state $s$, $K^s = \mathrm{true}$ and for any uncontrollable transition $(s, a, s') \in T, a \in A^u, [a](K^{sas'})$ always holds by hypothesis. ■

Notice that our composability notion differs from others existing in the literature (for example, Lin and Wonham, 1988) dealing with decentralized controller synthesis.

### *2.4. Priorities*

Priorities are widely used in modeling formalisms to restrict system behavior, especially for conflict resolution (Baeten et al., 1986; Cleaveland et al., 1996; Kwak et al., 1998). We introduce timed systems with priority constraints (Bornot et al., 2000; Bornot and Sifakis, 1997). Applying priorities amounts to restricting by a transition constraint which is a composable control invariant.

**Definition 2.9** (*Priorities*). A *priority order* is a strict partial order $\prec \subseteq A^c \times A$. We write $a_1 \prec a_2$ for $(a_1, a_2) \in \prec$.

A *priority rule* is a set of pairs $\mathrm{pr} = \{(C^j, \prec^j)\}_{j \in J}$, where $J$ is a finite index set, $\prec^j$ is a priority order, and $C^j$ is a state constraint that specifies when the priority order applies, such that for any $J' \subseteq J$ with $\bigwedge_{j \in J'} C^j \neq \mathrm{false}$, $\bigcup_{j \in J'} \prec^j$ is a priority order.

**Definition 2.10** *(Timed system with priorities).* A timed system TS with a priority rule pr as above, is the timed system $\mathrm{TS}/K_{\mathrm{pr}}$, where $K_{\mathrm{pr}}$ is the following transition constraint

$$K_{\mathrm{pr}} = \bigvee_{s \in S} s \wedge \bigwedge_{(C, \prec) \in \mathrm{pr}} \bigwedge_{a_j \in \mathrm{A}} [a_j] \left( \neg C(s) \vee \bigwedge_{\substack{i \in I \\ a_j \prec a_i}} \neg g_i \right)$$

$I$ denotes the set of indices such that $\{(s, a_i, s_i)\}_{i \in I}$ is the set of the transitions departing from $s$, and $h(s, a_i, s_i)$ is the tuple $(s, a_i, g_i, \tau_i, r_i, s_i)$ for each $i \in I$.

Notice that restriction by $K_{pr}$ transforms a controllable guard $g_j$ of transition $(s, a_j, s_j)$ of TS into

$$g'_j = g_j \wedge \bigwedge_{(C, \prec) \in \mathrm{pr}} \left( \neg C(s) \vee \bigwedge_{\substack{i \in I \\ a_j \prec a_i}} \neg g_i \right).$$

When the condition $C$ of a pair $(C, \prec) \in \mathrm{pr}$ is true, the corresponding priority order $\prec$ is applied. The application of this order results in allowing an action $a_j$ only if there is no enabled transition $a_i$ leaving $s$ that has priority over $a_j$.

**Lemma 2.3** *Let TS be a timed system and pr a priority rule. The constraint $K_{\mathrm{pr}}$ obtained when applying pr to TS as in definition 2.10, is a composable control invariant of TS.*

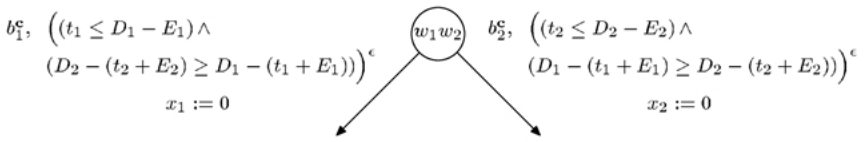


*Figure 3.* Applying $\mathrm{pr}_{\mathrm{llf}}$ on $\mathrm{TS}_{12}$ at state $(w_1 w_2)$.

**Proof:** $K_{\mathrm{pr}}$ is a transition constraint that only involves controllable actions by definition of priority order. Thus, by Proposition 2.4, it is a composable control invariant of TS. ■

**Example 2.3** Priorities can be used to resolve the conflict between actions $b_1$ and $b_2$ in the timed system $\mathrm{TS}_{12}$ of Example 2.2 from the control state $(w_1 w_2)$. The priority rule

$$\mathrm{pr}_{\mathrm{llf}} = \{(D_i - (t_i + E_i) < D_j - (t_j + E_j), b_j \prec b_i)\}_{(i,j) \in \{(1,2),(2,1)\}}$$

specifies a least laxity first policy (Mock, 1983) for conflict resolution. The expressions $D_i - (t_i + E_i)$ for $i = 1, 2$ represent the laxity of $P_i$. The restricted guards of $b_1$ and $b_2$ are shown in Figure 3.

## 3. Real-Time Application Modeling

We propose a methodology for modeling the real-time application to be scheduled as a set of communicating processes sharing common resources. The model is composed of two layers. The lower layer contains the application processes, and the upper layer handles resource management and synchronization.

The methodology provides essential guidelines for building abstractions of real-time applications relevant to scheduling.

### *3.1. Process Modeling*

We assume that the application is composed of a set of processes $P_i$ for $i \in \mathscr{I}$, where $\mathscr{I}$ is a finite index set. The processes share a set of resources $R$ partitioned into a set of preemptable resources $R_p$, and a set of non-preemptable resources $R_n$. Preemptable resources are resources from which a process can be preempted by another process, whereas a process using a non-preemptable resource keeps it until completion. Each process $P_i$ is represented by a timed system $TS_i = (S_i, A_i, T_i, X_i, \mathbf{b}_i, h_i)$ with the following attributes.

*States.* We define the following predicates on control states for a given process $P_i$, and a given resource $r \in R$:

1. $U_{i,r}(s)$ meaning that process $P_i$ uses $r$ at state $s \in S_i$.

2. $W_{i,r}(s)$ meaning that process $P_i$ waits for resource $r$ at state $s \in S_i$, and defined by $W_{i,r}(s) = \neg U_{i,r}(s) \wedge \exists (s, a, s') \in T_i \cdot U_{i,r}(s')$.

We use the abbreviations $U_i = \bigvee_{r \in R} U_{i,r}$, and $W_i = \bigvee_{r \in R} W_{i,r}$. The states $s$ that are neither waiting for, nor using any resource, i.e., $\neg(W_i \vee U_i)(s)$, are called *sleeping* states.

*Actions.* We distinguish the following types of actions for a process $P_i$:

1. *Arrival* actions leading from sleeping to waiting states.

2. *Begin* actions for some resource $r$ leading from a state $s$ such that $W_{i,r}(s)$ to a state $s'$ such that $U_{i,r}(s')$. $\mathrm{BGN}_{i,r}$ denotes this set of actions. Begin actions are usually eager to denote the fact that a process takes the resource as soon as it is allowed to do so.

3. *End* actions for some resource $r$, leading from a state $s$ such that $U_{i,r}(s)$ to a state $s'$ such that $\neg U_{i,r}(s')$. $\mathrm{END}_{i,r}$ denotes this set of actions.

Notice that the sets of arrival and begin actions are disjoint, as well as the sets of arrival and end actions.

The decision when to allocate a resource to a process is up to the scheduler, whereas the decision when to free a resource belongs to the process. Therefore, begin actions are considered as controllable, whereas end and arrival actions are uncontrollable. This means that the sets of begin and end actions are disjoint.

*Timing constraints.* We associate with each process $P_i$ the following timers:

1. A timer $t_i$ measuring the time elapsed since the arrival of $P_i$. Therefore, $t_i$ is reset by arrival actions and has its rate everywhere equal to one.

2. For each resource $r \in R$ used by $P_i$, a timer $x_{i,r}$ is introduced to measure execution time. This timer is reset at all begin actions acquiring $r$, and has its rate equal to 1 at states $s$ such that $U_{i,r}(s)$.

The timers are used to express the following classes of timing constraints.

*Inter-arrival constraints.* These constraints conjunct guards of arrival actions. Usually, they are conditions of the form $T_i^l \leq t_i \leq T_i^u$, where $T_i^l > 0$ and $T_i^u$ are lower and upper bounds of inter-arrival times. For process $P_i$ strictly periodic, take $T_i^l = T_i^u$, and for $P_i$ sporadic take $T_i^u = \infty$.

*Execution time constraints.* For a given resource $r \in R$ these constraints conjunct guards of actions issued from states such that $U_{i,r}(s)$, in particular actions of $\mathrm{END}_{i,r}$. They are

*Table 2*. Action classification.

| Action type | Most liberal guard | Resets | Ctrl. |
|---|---|---|---|
| Arrival of process $P_i$ | $T_i^l \leq t_i \leq T_i^u$ | $t_i$ | **u** |
| Begin of $P_i$ on $r$ | $t_i \leq D_{i,r} - E_{i,r}^u$ | $x_{i,r}$ | **c** |
| End of $P_i$ on $r$ | $E_{i,r}^l \leq x_{i,r} \leq E_{i,r}^u \wedge t_i \leq D_{i,r}$ | $\emptyset$ | **u** |

usually constraints of the form $E_{i,r}^l \leq x_{i,r} \leq E_{i,r}^u$ where $E_{i,r}^l > 0$ and $E_{i,r}^u$ are respectively the lower and the upper bounds of the execution time of $P_i$ on $r$.

*Deadline constraints*. These constraints express the fact that since the process arrival a resource has been used within a given deadline. They are usually expressed in terms of deadlines $D_{i,r}$ where $r$ is a resource used by process $P_i$. The resulting deadline constraints strengthen begin transitions on $r$ by condition $t_i \leq D_{i,r} - E_{i,r}^u$ and all the transitions issued from state $s$ such that $U_{i,r}(s)$ by $t_i \leq D_{i,r}$. Table 2 summarizes a classification of process actions with corresponding timing constraints.

A timed system of processes is specified by a set of processes $P_i$, sets of resources $R_p$ and $R_n$, and the associated predicates $U_{i,r}$.

### *3.2. The Synchronization Layer*

To model a real-time application, we adopt the architecture shown in Figure 4, which suggests a decomposition and an incremental description principle.

The real-time application model is built from the timed systems representing the processes, by successive application of a synchronization layer and of a scheduler. The synchronization layer ensures mutual exclusion on shared resources, and provides a

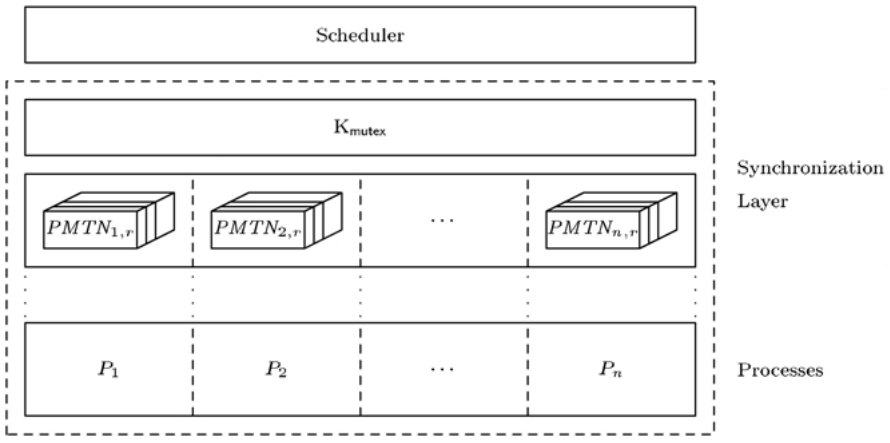


*Figure 4*. Architecture of the global model.

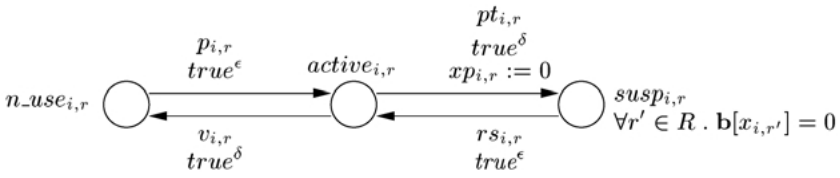


*Figure 5.* The preemption management system $\text{PMTN}_{i,r}$.

preemption mechanism. It is in charge of guaranteeing only functional properties of the application.

The scheduler interacts with the underlying system by resolving non-determinism so as to satisfy non-functional properties, schedulability requirements in particular.

To model preemption mechanisms, we use for each process $P_i$ and each preemptable resource $r \in R_p$ a timed system $\text{PMTN}_{i,r}$ (see Figure 5). The latter has three control states $\text{n\_use}_{i,r}$, $\text{active}_{i,r}$ and $\text{susp}_{i,r}$ meaning respectively, that the process $P_i$ is not using $r$, is effectively using $r$, is suspended due to preemption from $r$. The synchronization of $\text{PMTN}_{i,r}$ and $P_i$ is defined so as to agree with this meaning, i.e., $P_i$ is at a state $s$ such that $\neg U_{i,r}(s)$ if and only if $\text{PMTN}_{i,r}$ is at state $\text{n\_use}_{i,r}$.

Following notation from semaphores, the actions $p_{i,r}$ and $v_{i,r}$ represent granting $r$ to $P_i$ and freeing of $r$ by $P_i$, respectively. The actions $pt_{i,r}$ and $rs_{i,r}$ represent $P_i$ getting preempted from $r$ and resuming $r$, respectively.

When a process $P_i$ is suspended due to preemption from some resource $r$, all the execution timers of $P_i$ are stopped:

$$\forall r \in R \cdot \mathbf{b}_{\text{susp}_{i,r}}[x_{i,r}] = 0$$

We suppose that any begin (resp. end) action acquiring (resp. freeing) a preemptable resource $r$ acquires (resp. frees) only $r$.

The synchronization between processes and $\text{PMTN}_{i,r}$ is specified by the set of synchronization actions $\Sigma$:

$$\Sigma = \{b_i \mid p_{i,r},\ b_i \mid p_{i,r} \mid pt_{j,r},\ e_i \mid v_{i,r} \mid i,\ j \in \mathcal{I} \wedge i \neq j \wedge r \in R_p \wedge b_i \in \text{BGN}_{i,r} \wedge e_i \in \text{END}_{i,r}\}$$

Intuitively, each begin action $b_i$ of some process $P_i$ acquiring a preemptable resource $r$ synchronizes with an allocating action $p_{i,r}$ and, if there is already a process using $r$, with the preempting action $pt_{j,r}$. End actions $e_i$ synchronize with freeing actions $v_{i,r}$ of $\text{PMTN}_{i,r}$.

Mutual exclusion on the resources in $R$ is expressed by the constraint

$$K_{\text{mutex}} = \bigwedge_{\substack{i,j \in \mathcal{I} \\ i \neq j}} \left( \bigwedge_{r_n \in R_n} \neg(U_{i,r_n} \wedge U_{j,r_n}) \wedge \bigwedge_{r_p \in R_p} \neg(\text{active}_{i,r_p} \wedge \text{active}_{j,r_p}) \right)$$

$K_{\text{mutex}}$ says that for any pairs of distinct processes $P_i$ and $P_j$, it is not possible to use at some state the same non-preemptable resource $r_n$ or preemptable resource $r_p$.

The timed system of processes to be scheduled TS is obtained by composing the processes and restricting the global system according to the steps described above. Formally,

$$\mathrm{TS} = (\|_{\Sigma}\ \{\mathrm{TS}_i \mid i \in \mathscr{I}\} \cup \{\mathrm{PMTN}_{i,r} \mid i \in \mathscr{I} \wedge r \in R_p\})/K_{\mathrm{mutex}}$$

Let $\mathrm{AL}_{i,r}$ denote the actions of TS allocating $r \in R$ to $P_i$: if $r$ is not preemptable, $\mathrm{AL}_{i,r} = \mathrm{BGN}_{i,r}$; if $r$ is preemptable, $\mathrm{AL}_{i,r} = \{b_i \mid p_{i,r}, b_i \mid p_{i,r} \mid pt_{j,r}, rs_{i,r} \mid b_i \in \mathrm{BGN}_{i,r} \wedge j \in \mathscr{I}\}$.

The following section discusses how TS can be further restricted until a scheduled system is obtained.

## 4. Specification of Scheduling Policies

### *4.1. Methodology*

Many existing scheduling policies distinguish on the one hand rules for resolving conflicts on processes by assigning process priorities, and on the other hand admission control rules for deciding whether some process is eligible for resource allocation (Keshav, 1997). For example, the priority ceiling protocol (Sha et al., 1990) schedules the process with the highest current priority among the processes that are waiting for the processor, whereas a process $P_i$ is eligible for the allocation of a free resource if the current priority of $P_i$ is higher than the priority ceilings of all resources currently allocated to processes other than $P_i$.

We formalize this decomposition of a scheduling policy as a conjunction of two transition constraints: $K_{\mathrm{pol}} = K_{\mathrm{adm}} \wedge K_{\mathrm{res}}$, where $K_{\mathrm{adm}}$ is an admission control constraint specifying eligibility for resource allocation, whereas $K_{\mathrm{res}}$ is a constraint specifying how conflicts between two or more processes waiting for the same resource are resolved.

In principle, for each resource, different admission control and conflict resolution constraints are applicable. We take $K_{\mathrm{adm}} = \wedge_{r \in R} K^r_{\mathrm{adm}}$ and $K_{\mathrm{res}} = \wedge_{r \in R} K^r_{\mathrm{res}}$. For example, resources that are crucial for the correct functioning of a system could be allocated according to a conservative policy guaranteeing deadlock-freedom, whereas resources shared by less critical processes could be managed using a more optimistic policy in order to use them more efficiently. In practice, care should be taken to apply compatible policies so as to avoid inconsistency (Bornot et al., 2000).

#### *4.1.1. Admission Control*

The constraint $K_{\mathrm{adm}} = \wedge_{r \in R} K^r_{\mathrm{adm}}$ restricting resource allocations is specified by transition constraints of the form

$$K^r_{\mathrm{adm}} = \bigvee_{s \in S} s \wedge \bigwedge_{\substack{(s,a,s') \in \mathrm{T} \\ \exists i.a \in \mathrm{AL}_{i,r}}} [a](K^{sas'})$$

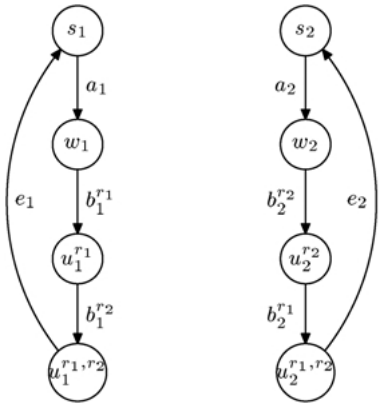


*Figure 6.* Two processes sharing two resources.

The $X$-constraint $K^{sas'}$ restricts all the actions $a$ that label transitions $(s, a, s')$ allocating $r$ to some process $P_i$.

**Example 4.1** Consider the system of two processes $P_1$ and $P_2$ as in Figure 6, where two processes share two non-preemptable resources. We assume that mutual exclusion has been ensured by restricting with $K_{\text{mutex}} = \neg(u_1^{r_1} \wedge u_2^{r_1,r_2}) \wedge \neg(u_1^{r_1,r_2} \wedge u_2^{r_2}) \wedge \neg(u_1^{r_1,r_2} \wedge u_2^{r_1,r_2})$. A deadlock arises when $P_1$ has obtained $r_1$ and is in $u_1^{r_1}$ waiting for $r_2$, and $P_2$ has obtained $r_2$ and is in $u_2^{r_2}$ waiting for $r_1$. This situation can be avoided by the following admission control: if $P_1$ holds $r_1$, then $P_2$ is prevented from acquiring $r_2$; if $P_2$ holds $r_2$, then $P_1$ is prevented from acquiring $r_1$. The constraints

$$K_{\text{adm}}^{r_1} = (u_2^{r_2} \vee u_2^{r_1,r_2}) \Rightarrow [b_1^{r_1}](\text{false})$$
$$K_{\text{adm}}^{r_2} = (u_1^{r_1} \vee u_1^{r_1,r_2}) \Rightarrow [b_2^{r_2}](\text{false})$$

represent the admission control policy for a process to obtain $r_1$ and $r_2$, respectively.

**Example 4.2** A non-idling scheduling policy for some resource $r$ is a policy where a free resource $r$ is granted as soon as it is requested, i.e., it is not possible to delay the allocation of a requested free resource. Non-idling scheduling for $r$ can be specified by a constraint of the form

$$K_{\text{n\_idle}}^{r} = \bigwedge_{i \in \mathscr{I}} \bigwedge_{a \in \text{AL}_{i,r}} [a](\text{asap}_r = 0)$$

where $\text{asap}_r$ is an additional timer reset at any request action where some process $P_i$ enters a state $s$ such that $W_{i,r}(s)$, and at any action of $\text{END}_{i,r}$ where $P_i$ frees $r$. The timer $\text{asap}_r$ measures the minimum time elapsed since $r$ has been freed or the last request of $r$ by a process $P_i$. The requirement that $\text{asap}_r = 0$ when $r$ is allocated means that the

resource has just been freed, or some process has just started waiting. The restriction of allocation action guards by $\text{asap}_r = 0$ imposes non-idling for resource $r$.

### 4.1.2. *Conflict Resolution*

We assume that for each resource $r$, conflicts are resolved according to a partial order on the set of processes $\{P_i \mid i \in \mathcal{I}\}$. The partial order is specified as a set of state constraints $\{\mathscr{C}^r_{ij}\}_{i,j \in \mathcal{I}}$, such that $\mathscr{C}^r_{ij} \wedge \mathscr{C}^r_{jk} \Rightarrow \mathscr{C}^r_{ik}$. When $\mathscr{C}^r_{ij}$ holds, process $P_j$ has priority over process $P_i$ for using resource $r$. Notice that $\mathscr{C}^r_{ij}$ may depend on timer values as well as on control states. Two cases of conflict may appear.

First, if two processes $P_i$ and $P_j$ are waiting for $r$, actions of $\text{AL}_{j,r}$ are given priority over actions of $\text{AL}_{i,r}$ by the priority rule

$$\text{pr}^r = \{(\mathscr{C}^r_{ij},\ \text{AL}_{i,r} \prec \text{AL}_{j,r})\}_{i,j \in \mathcal{I}}$$

where for $B, C \subseteq A,\ B \prec C \iff \forall (b,c) \in B \times C \cdot b \prec c$. Notice that transitivity of $\prec$ is guaranteed by the property $\mathscr{C}^r_{ij} \wedge \mathscr{C}^r_{jk} \Rightarrow \mathscr{C}^r_{ik}$.

Second, in case of conflict between a process $P_j$ using the resource, and a process $P_i$ attempting to preempt it, preemption must be prevented unless $P_i$ is of higher priority. Preemption by lower or equal priority processes is prevented by restricting with the constraint

$$K^r_{\text{res}-\text{pt}} = \bigvee_{s \in \text{S}} s \wedge \bigwedge_{\substack{i \neq j \\ active_{j,r}(s)}} \bigwedge_{a \in AL_{i,r}} [a](\mathscr{C}^r_{ji}(s))$$

Intuitively, if some $P_j$ is active on $r$, actions of $P_i$ preempting $P_j$ from $r$ are enabled only if $\mathscr{C}^r_{ji}$.

The constraint $K^r_{\text{res}}$ specifying arbitration between processes according to the given order is obtained as a conjunction $K_{\text{pr}^r} \wedge K^r_{\text{res}-\text{pt}}$, where $K_{\text{pr}^r}$ is the constraint corresponding to the priority rule $\text{pr}^r$.

### 4.1.3. *Getting the Scheduled System*

The scheduled system TS′ obtained by application of a scheduling policy on TS is of the form

$$\text{TS}' = \text{TS} \Big/ \bigwedge_{r \in R} \left(K^r_{\text{adm}} \wedge K^r_{\text{res}}\right)$$

It is the restriction of TS by the admission control and conflict resolution constraints for all the resources of the system.

### *4.2. Specifying Well-Known Policies*

We define some scheduling policies on the timed system TS composed of the set of processes $\{P_i \mid i \in \mathscr{I}\}, \mathscr{I} = \{1, \ldots, n\}$, as in the previous paragraph. For states where the policy does not specify a total order, we complete the policy order by the static order: $P_j$ has priority over $P_i$ if $j < i$.

*The fifo policy.* A scheduler follows a first in first out policy (fifo) on the resource $r$ if it is granted to the process that has been waiting for the longest time. The fifo policy is specified by

$$\mathscr{C}^r_{ij} = (t_i < t_j) \vee (t_i = t_j \wedge j < i)$$

The first term of $\mathscr{C}^r_{ij}$ means that whenever two processes $P_i$ and $P_j$ are both waiting for $r$, $P_j$ is served prior to $P_i$ if process $P_j$ has been waiting for longer time than process $P_i$, i.e., $t_i < t_j$. The second term ensures a strict allocation order in case of conflict (i.e., when $t_i = t_j$).

*The edf policy.* A scheduler follows an edf policy on the resource $r$ if it is granted to the waiting process that is closest to its relative deadline (Liu and Layland, 1973). The edf policy is specified by

$$\mathscr{C}^r_{ij} = (D_{j,r} - t_j < D_{i,r} - t_i) \vee (D_{j,r} - t_j = D_{i,r} - t_i \wedge j < i)$$

i.e., whenever there are two processes $P_i$ and $P_j$ waiting for $r$, the actions granting $r$ to $P_j$ have immediate priority over the actions granting $r$ to $P_i$ if $P_j$ is closer to its relative deadline than $P_i$ (namely, $D_{j,r} - t_j < D_{i,r} - t_i$).

*The rms policy.* The algorithm of preemptive rate-monotonic scheduling (rms, Liu and Layland, 1973) assigns to each strictly periodic process a fixed priority such that processes with shorter period have higher priority, i.e., if $T_i > T_j$, $P_j$ has priority over $P_i$. The rms policy is specified exactly as the edf policy by replacing $D_{j,r} - t_j < D_{i,r} - t_i$ by the condition $T_j < T_i$.

*The priority ceiling protocol.* According to the priority ceiling protocol (Sha et al., 1990), we consider a system composed of processes sharing a processor CPU, and a set of non preemptable resources $R_n$. The processor is the only preemptable resource, i.e., $R_p = \{\text{CPU}\}$. The priority ceiling protocol specifies that:

1. The current priority of a process is the maximum of its own fixed priority, and the priorities of the processes it blocks;

2. A process can only obtain a resource $r$ of ceiling $c_r$ if its current priority is higher than the ceiling of any resource allocated to some other process;

3. The process with the highest current priority is given the processor.

We give here only a sketch of how the priority ceiling protocol can be modeled in this framework, and omit some more technical details for sake of simplicity.

We suppose that the process indices correspond to the process priorities such that a process of lower index has a higher priority. For $r \in R_n$, let $c_r = \min\{i \mid i \in \mathscr{I} \wedge \exists s \in S_i \cdot U_{i,r}(s)\}$ be the priority ceiling of $r$, i.e., the index of the highest-priority process that may use $r$. The predicate

$$\begin{aligned}&\text{blocking}_{ij}(s)\\ &\quad = i \neq j \wedge \exists k \in \mathscr{I} \cdot \exists r_1,\ r_2 \in R_n \cdot c_{r_1} \leq k \leq j \wedge U_{i,r_1}(s) \wedge W_{k,r_2}(s)\end{aligned}$$

characterizes the states $s \in S$ where a given process $P_j$ is not allowed to obtain a non-preemptable resource because some process $P_i$ holds a non-preemptable resource $r_1$ of ceiling $c_{r_1} \leq j$, and there is some process $P_k$ of intermediate priority waiting for a resource $r_2$. It can be shown that (2) is modeled by the admission control policy

$$K^r_{\text{adm}} = \bigwedge_{s \in S} \bigwedge_{\substack{i,j \in \mathscr{I} \\ \text{blocking}_{ij}(s)}} \left( s \Rightarrow \bigwedge_{a \in \text{AL}_{j,r}} [a](\text{false}) \right)$$

for any $r \in R_n$. Conflicts concerning the allocation of the processor are resolved according to (3) by the conflict resolution policy

$$\mathscr{C}^{\text{CPU}}_{ij} = i > j \wedge \neg\text{blocking}_{ij} \vee \text{blocking}_{ji}$$

for the processor. A process $P_j$ is given priority over another process $P_i$ if $P_j$ is of higher priority than $P_i$ and not blocked by $P_i$, and in states $s$ where $\text{blocking}_{ji}(s)$ holds.

**Example 4.3** Consider three identical periodic processes $P_1$, $P_2$, $P_3$, as depicted in Figure 7. Timing constraints have been omitted. The processes use the processor CPU and one shared non-preemptable resource $r$ in the following states: $U_{i,\text{CPU}} = u_i \vee u^r_i$, and $U_{i,r} = u^r_i$ for $i \in \{1,2,3\}$. The predicates defined above become

$$\begin{aligned}&\text{blocking}_{12} = u^r_1 \wedge w^r_2 && \text{blocking}_{21} = u^r_2 \wedge w^r_1\\ &\text{blocking}_{13} = u^r_1 \wedge (w^r_2 \vee w^r_3) && \text{blocking}_{31} = u^r_3 \wedge w^r_1\\ &\text{blocking}_{23} = u^r_2 \wedge (w^r_1 \vee w^r_3) && \text{blocking}_{32} = u^r_3 \wedge (w^r_1 \vee w^r_2)\end{aligned}$$

$$\begin{aligned}K^r_{\text{adm}} = {} & (u^r_2 \vee u^r_3) \wedge w^r_1 \Rightarrow [b^r_1](\text{false}) \wedge\\ & u^r_1 \wedge w^r_2 \vee u^r_3 \wedge (w^r_1 \vee w^r_2) \Rightarrow [b^r_2](\text{false}) \wedge\\ & u^r_1 \wedge (w^r_2 \vee w^r_3) \vee u^r_2 \wedge (w^r_1 \vee w^r_3) \Rightarrow [b^r_3](\text{false})\end{aligned}$$

$$\begin{aligned}&\mathscr{C}^{\text{CPU}}_{12} = u^r_2 \wedge w^r_1 && \mathscr{C}^{\text{CPU}}_{21} = \neg(u^r_2 \wedge w^r_1)\\ &\mathscr{C}^{\text{CPU}}_{13} = u^r_3 \wedge w^r_1 && \mathscr{C}^{\text{CPU}}_{31} = \neg(u^r_3 \wedge w^r_1)\\ &\mathscr{C}^{\text{CPU}}_{23} = u^r_3 \wedge (w^r_1 \vee w^r_2) && \mathscr{C}^{\text{CPU}}_{32} = \neg(u^r_3 \wedge (w^r_1 \vee w^r_2))\end{aligned}$$

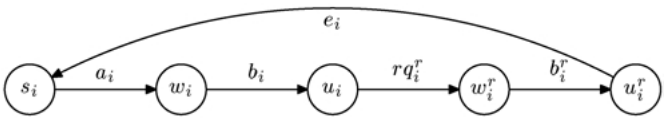


*Figure 7.* The process $P_i$ using the processor and a resource $r$.

Since there is only one non-preemptable resource $r$, $K^r_{\text{adm}}$ only expresses mutual exclusion on $r$, according to (2). The priority rule defined by the $\mathscr{C}^{\text{CPU}}_{ij}$ gives priority to the process with lower index, unless it blocks on $r$. Notice $\mathscr{C}^{\text{CPU}}_{23}$: $P_3$ has priority over $P_2$ in particular if $P_3$ is using $r$, and $P_1$ is waiting for $r$, so as to prevent priority inversion (Sha et al., 1990).

## 5. Schedulability Requirements and Analysis

This section presents the last step of the modeling methodology. We define schedulability requirements as a constraint $K_{\text{sched}}$ expressing the fact that each application process meets its timing constraints. Given the model of a real-time application scheduled by some policy, it is necessary to check its schedulability by searching for a control invariant which implies $K_{\text{sched}}$. This requires, in principle, the use of a timing analysis tool. We provide results allowing simplification of the schedulability analysis task.

### *5.1. Schedulability Requirements*

Schedulability requirements $K_{\text{sched}}$ express the fact that the timing constraints of the application processes (defined in 3.1) are met. This practically means that for each process, inter-arrival, execution time and deadline constraints are never violated. As these constraints strengthen action guards, the non-violation of timing constraints (in particular, of their upper bounds) implies absence of deadlock, i.e., the possibility to execute an action from any state satisfying $K_{\text{sched}}$. This leads to the formulation of schedulability requirements as a conjunction $K_{\text{sched}} = \bigwedge_{i \in \mathscr{I}} K^i_{\text{sched}}$ of the schedulability requirements of the individual processes.

**Definition 5.1** ($\Diamond$). Let $C$ be an $X$-constraint, and $s \in S$ a control state. We use the notation for all $\mathbf{x} \in \mathbb{N}^m$,

$$(\Diamond_s C)(\mathbf{x}) = \exists t \geq 0 \cdot C(\mathbf{x} + t\mathbf{b}_s)$$

Notice that $(\Diamond_s C)(\mathbf{x})$ means ''eventually $C(\mathbf{x})$''; if time can progress indefinitely at control state $s$.

Given a process $P_i$, $i \in \mathcal{I} = \{1, \ldots, n\}$, the associated schedulability requirement $K^i_{\text{sched}}$ is defined by the formula

$$K^i_{\text{sched}} = \bigvee_{s=(s_1,\ldots,s_n)\in S} s_i \wedge \left( \bigvee_{(s_i,a,s'_i)\in T_i} \Diamond_s g_a \right)$$

where $g_a$ is the guard of the transition $(s_i, a, s'_i)$.

The invariance of $K^i_{\text{sched}}$ expresses the fact that it is always possible to execute some action. From some state $(s, \mathbf{x})$ satisfying $K^i_{\text{sched}}$ there exists some time $t$ such that $(s, \mathbf{x} + t\mathbf{b}_s)$ satisfies the guard of an action exiting from $s_i$ since by the semantics of timed systems (Definition 2.2), time progress cannot stop as long as no guard is true. It follows that inevitably some action of $P_i$ exiting from $s$ will be enabled. Thus, if $K^i_{\text{sched}}$ is an invariant then the process $P_i$ is deadlock-free (i.e., it is always possible to execute some action).

If all the guards of a process are either eager or delayable with an upper bound, the formula $K^i_{\text{sched}}$ expresses a liveness property, i.e., always eventually some action is executed. In that case, from the definition of $\varphi$ (see Definition 2.2), $\Diamond_s g_a$ implies that eventually a state is reached from which time progress stops and then an action must be taken by its deadline.

Notice that in process modeling of Section 3.1, the guards of the transitions exiting waiting or executing states are either eager or delayable with an upper bound. If a process is not blocked at waiting or using states, then the deadlines are respected (see Table 2). $K^i_{\text{sched}}$ precisely guarantees that begin and end actions will eventually take place.

**Example 5.1** Let us consider the single periodic process described by Figure 2. The associated schedulability requirement is

$$K_{\text{sched}} = w \wedge t \leq D - E \vee e \wedge x \leq E \wedge t \leq D \vee s \wedge t \leq T$$

It is easy to check that $K_{\text{sched}}$ is a proper invariant. As the transition guards have upper bounds, $K_{\text{sched}}$ guarantees not only deadlock freedom but also liveness. If the process is initialized in $K_{\text{sched}}$ then always eventually some action happens, and thus the timing constraints are respected.

### *5.2. Getting a Correctly Scheduled System*

In this paragraph, we show how the presented results can be used to simplify schedulability analysis and eventually get a correctly scheduled system.

A thesis of the paper is that scheduling requirements are the conjunction $K_{\text{sched}} \wedge K_{\text{pol}}$ where $K_{\text{sched}}$ are schedulability requirements about process dynamics, and $K_{\text{pol}}$ are policy requirements about resource management. As already explained, solving the scheduling problem for a system TS amounts to finding a non-empty control invariant $K$ such that $K \Rightarrow K_{\text{sched}} \wedge K_{\text{pol}}$.

The following proposition allows a drastic simplification of this problem.

**Proposition 5.1** Given a timed system TS and some scheduling requirements $K_{\text{sched}} \wedge K_{\text{pol}}$, if $K_S$ is a control invariant of $\text{TS}/K_{\text{pol}}$ such that $K_S \Rightarrow K_{\text{sched}}$, then $K = K_{\text{pol}} \wedge K_S$ is a control invariant of TS such that $K \Rightarrow K_{\text{pol}} \wedge K_{\text{sched}}$. Conversely, if $K$ is a non-empty control invariant of TS, such that $K \Rightarrow K_{\text{pol}} \wedge K_{\text{sched}}$ then there exists a non-empty control invariant $K_S$ of $\text{TS}/K_{\text{pol}}$ such that $K_S \Rightarrow K_{\text{sched}}$.

**Proof:** If $K_S \Rightarrow K_{\text{sched}}$ and $K_S$ is a control invariant of $\text{TS}/K_{\text{pol}}$, then $(\text{TS}/K_{\text{pol}})/K_S \models \text{inv}(K_S)$. As $K_{\text{pol}}$ is a composable control invariant we have that $\text{TS}/(K_{\text{pol}} \wedge K_S) \models \text{inv}(K_{\text{pol}} \wedge K_S)$. Conversely, if $K$ such that $K \Rightarrow K_{\text{pol}} \wedge K_{\text{sched}}$ is a non-empty control invariant of TS, take $K_S = K$. We have $\text{TS}/(K_{\text{pol}} \wedge K_S) = \text{TS}/(K_{\text{pol}} \wedge K) = \text{TS}/K \models \text{inv}(K)$ which is true by hypothesis. ■

This proposition provides a justification of the proposed modeling methodology. Instead of searching for control invariants of TS satisfying $K_{\text{sched}} \wedge K_{\text{pol}}$, we first restrict TS by $K_{\text{pol}}$ and perform schedulability analysis on the restricted system.

Current approaches to scheduling can be summarized as follows, in the light of the presented approach.

A first approach consists in using analytical methods to find criteria ensuring schedulability of $\text{TS}/K_{\text{pol}}$ for particular scheduling policies such as edf, rms, llf. Available results are applicable when the processes and their timing constraints meet specific conditions about periods, execution times, deadlines, resource allocation, etc. In this approach, schedulability analysis deals essentially with checking that the system meets the schedulability criteria prescribed by the theory. It does not require the use of a model representing the dynamic behavior of the system to be scheduled. Current engineering practice essentially adopts this approach, for example, Härbour et al. (1993) and Vestal (1994).

A second approach (Ben-Abdallah et al., 1998; Ericsson et al., 1999) consists in building explicitly a model TS of the system to be scheduled, i.e., the application processes with the synchronization layer, and finding a non-empty control invariant $K \Rightarrow K_{\text{sched}}$ without considering particular scheduling policies. This is the most general approach, but the algorithmic method for computing control invariants is of prohibitive complexity. For this reason, sometimes, the existence of a proper invariant implying $K_{\text{sched}}$ is explored (Ben-Abdallah et al., 1999): this is a sufficient condition for schedulability requiring techniques of lower complexity which do not distinguish between controllable and uncontrollable actions.

The presented methodology provides a framework for combining the two approaches. The use of scheduling policies in the second approach can make the schedulability analysis problem more tractable. The application of a scheduling policy $K_{\text{pol}}$ to a system TS reduces the non-determinism of TS and consequently the complexity of its state space. So, in principle, it is easier to compute control invariants of $\text{TS}/K_{\text{pol}}$ which imply $K_{\text{sched}}$ than control invariants of the non-restricted system TS. However, $K_{\text{pol}}$ should be chosen so that $K_{\text{pol}} \wedge K_{\text{sched}}$ contains at least one non-empty control invariant and this cannot be guaranteed by existing results.

We provide a method for scheduling a timed system TS with given schedulability requirements, by successive applications of scheduling policies. Its application requires

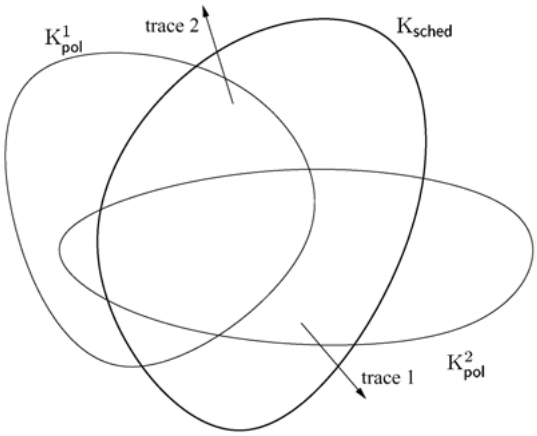


*Figure 8.* Policies chosen according to diagnostics.

the use of a timing analysis tool allowing to verify if a constraint is a control invariant and providing diagnostics in case of negative answer. Diagnostics are given in the form of traces starting with a controllable transition from a state which satisfies the constraint and leading to a state where the constraint is false. They can provide guidance for choosing $K_{\text{pol}}$. The idea is that $K_{\text{pol}}$ is used to restrict the behaviors violating $K_{\text{sched}}$ as illustrated in Figure 8. $K_{\text{pol}}$ is the conjunction of $K^1_{\text{pol}}$ and $K^2_{\text{pol}}$ used to eliminate trace 1 and trace 2, respectively.

The method can be specified as follows.

$K := K_{\text{sched}}$;
<u>while</u> $\neg(\text{TS}/K \models \text{inv}(K))$<u>do</u>
  <u>choose</u> $K_{\text{pol}}$;  $K := K \wedge K_{\text{pol}}$
<u>od</u>

The following example (Section 6) illustrates this method. We use PROMETHEUS (Gößler, 2001), a scheduler design tool developed at VERIMAG, to build the model of the system to be scheduled. In the iterative process, the timing analysis tool KRONOS is used to check that the restricted system satisfies $K_{\text{sched}}$. In case of non satisfaction, KRONOS provides diagnostics in the form of traces explaining why $K_{\text{sched}}$ is violated.

## 6. Example: An Active Structural Control System

We consider an embedded active structural control system, used to prevent damages on buildings caused by earthquakes or strong wind (Braberman, 2000; Elseaidy et al., 1977;

Kang et al., 1996). It is composed of three processes. A cyclic process called *modeler* makes a computation which takes between 2.3 and 2.5 ms and updates a shared buffer, which takes 1 ms. A process of period between 3.2 and 14.5 ms called *pulser*, reads the data stored in the buffer and writes them to actuators. Reading takes 0.2 ms, writing 5.8 ms. Both modeler and pulser share the same processor.

A further timing constraint to be respected requires that the data read by the pulser be fresh, i.e., have been stored by the modeler at most 13 ms ago. Freshness is imposed by a process called *data*.

The three processes are modeled as in Figure 9, where a time unit corresponds to 0.1 ms.

The schedulability requirement is $K_{\text{sched}} = K_{\text{sched}}^{\text{modeler}} \wedge K_{\text{sched}}^{\text{data}} \wedge K_{\text{sched}}^{\text{pulser}}$ with

$$
\begin{aligned}
K_{\text{sched}}^{\text{modeler}} &= \mathit{sleeping}_m \vee \mathit{waiting\_c} \vee (\mathit{computing} \wedge x_m \leq 25) \\
&\quad \vee\ \mathit{waiting\_u} \vee (\mathit{updating} \wedge x_m \leq 10) \\
K_{\text{sched}}^{\text{data}} &= \text{true} \\
K_{\text{sched}}^{\text{pulser}} &= (\mathit{sleeping}_p \wedge t_p \leq 145) \vee (\mathit{waiting\_r} \wedge t_p \leq 85) \\
&\quad \vee\ (\mathit{reading} \wedge x_p \leq 2 \wedge t_p \leq 87 \wedge t_p - x_p \leq 85) \\
&\quad \vee\ (\mathit{writing} \wedge x_p \leq 58 \wedge t_p \leq 145 \wedge t_p - x_p \leq 87)
\end{aligned}
$$

It is easy to see that $K_{\text{sched}}^{\text{modeler}}$, $K_{\text{sched}}^{\text{data}}$, and $K_{\text{sched}}^{\text{pulser}}$ are proper invariants of the processes *modeler*, *data*, and *pulser*, respectively.

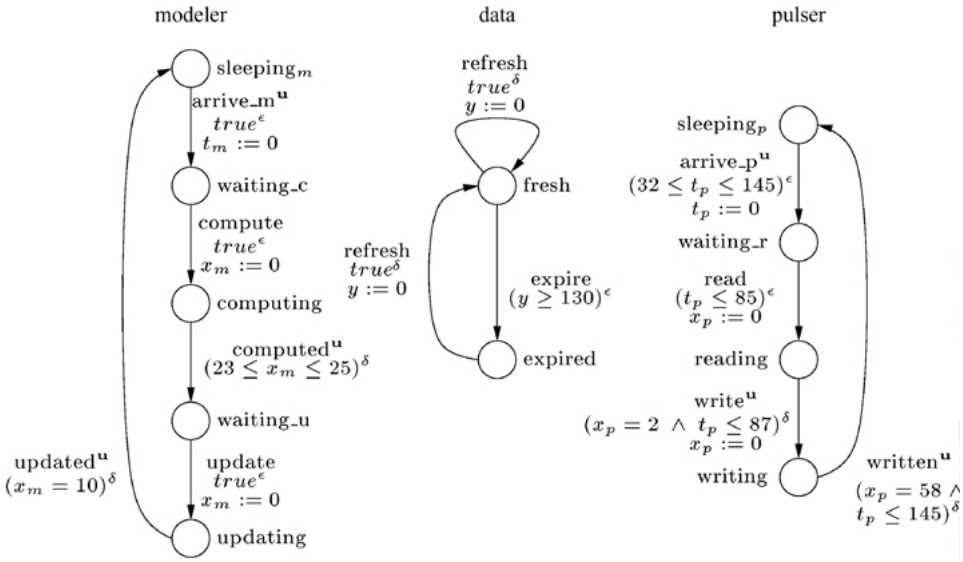


*Figure 9.* The active structural control system.

The three processes share two resources—the processor *CPU*, and the *buffer*—that are used in the following states:

$$\begin{aligned} U_{modeler,CPU} &= computing \vee updating,\\ U_{modeler,buffer} &= updating,\\ U_{pulser,CPU} &= reading \vee writing,\\ U_{pulser,buffer} &= reading \end{aligned}$$

We model the freshness constraint saying that *pulser* must not enter the state *reading* when *data* has expired, by introducing a third resource called *data_exp* with $U_{data,data\_exp} = expired$ and $U_{pulser,data\_exp} = reading$. Mutual exclusion on the use of *data_exp* ensures that *pulser* blocks on *data_exp* if the state of *data* is *expired*.

In the composition of the three processes, the action *updated* synchronizes with refresh, which means that at the instant where the modeler finishes writing data to the buffer, *data* changes to (or remains in) the *fresh* state. Formally, $\Sigma = \{updated \mid refresh\}$.

Mutual exclusion for the three resources is ensured by

$$\begin{aligned} K_{\text{mutex}} = \neg\big((computing \vee updating) \wedge (reading \vee writing)\big)\\ \wedge \neg(expired \wedge reading) \end{aligned}$$

Table 3 shows how the guards of the controllable actions of $\text{TS} = \|_{\Sigma}\{modeler, data, pulser\}$ are progressively restricted by the constraints.

The process *data* changes from *fresh* to *expired* as soon as its lifetime reaches 13 ms, provided that the process *pulser* has left the critical section *reading*.

In order to give the action *expire* priority over an entry of *pulser* into *reading*, we apply the conflict resolution policy $pr^{data\_exp} = \{(fresh \wedge waiting\_r,\ \{read \prec expire\})\}$ to get TS′:

$$\text{TS}' = (\|_{\Sigma}\ \{modeler, data, pulser\})/(K_{\text{mutex}} \wedge\ K_{pr^{data\_exp}})$$

where $K_{\text{pr}^{\text{data\_exp}}}$ is the constraint corresponding to the priority rule $\text{pr}^{data\_exp}$. The constraint $K_{\text{sched}} = K^{modeler}_{\text{sched}} \wedge K^{data}_{\text{sched}} \wedge K^{pulser}_{\text{sched}}$ is not a proper invariant of TS′. For example, the

*Table 3*. Successive restrictions of the guards of controllable actions.

| | TS | $/(K_{\text{mutex}} \wedge K_{\text{pr}^{\text{data\_exp}}})$ | $/K_{\text{pr}^{\text{CPU}}}$ |
|---|---|---|---|
| compute | *true* | $\neg(reading \vee writing)$ | $\neg(reading \vee writing \vee fresh \wedge waiting\_r \wedge t_p \leq 85 \wedge y < 130)$ |
| update | *true* | $\neg(reading \vee writing)$ | $\neg(reading \vee writing \vee fresh \wedge waiting\_r \wedge t_p \leq 85 \wedge y < 130)$ |
| expire | $y \geq 130$ | $y \geq 130 \wedge \neg reading$ | $y \geq 130 \wedge \neg reading$ |
| read | $t_p \leq 85$ | $t_p \leq 85 \wedge \neg(computing \vee updating \vee expired \vee y \geq 130)$ | $t_p \leq 85 \wedge \neg(computing \vee updating \vee expired \vee y \geq 130)$ |

process *modeler* can be scheduled five times before giving the process *pulser* a turn. Hereafter, *pulser* will get deadlocked in *waiting_r*.

Let us now restrict $\mathrm{TS}'$ by giving priority to *pulser* for access to the processor if *data* is *fresh*:

$$pr^{\mathrm{CPU}} = \{(fresh,\ \{compute \prec read,\ update \prec read\})\}$$

$K_{\mathrm{sched}}$ is a proper invariant of $\mathrm{TS}'/K_{pr^{CPU}}$, as can be verified using KRONOS. In fact, as soon as *pulser* blocks on expired *data*, its deadline is far enough for *modeler* to update the buffer before *pulser* misses its deadline. The last column of Table 3 specifies the scheduler maintaining the schedulability requirement.

Let us now slightly modify this example. We strengthen the timing constraints and require a maximal inter-arrival time of 200 ms for *modeler*. This is modeled by replacing the guard $true^{\varepsilon}$ of the action $arrive\_m^{\mathbf{u}}$ by $(t_m \leq 200)^{\varepsilon}$ in Figure 9, and strengthening accordingly the guards of the controllable actions of this new process $modeler_2$. The schedulability requirement is now

$$\begin{aligned} K_{\mathrm{sched}}^{modeler_2} = {} & (sleeping_m \wedge t_m \leq 200) \vee (waiting\_c \wedge t_m \leq 165) \\ & \vee (computing \wedge x_m \leq 25 \wedge t_m \leq 190 \wedge t_m - x_m \leq 165) \\ & \vee (waiting\_u \wedge t_m \leq 190) \\ & \vee (updating \wedge t_m \leq 200 \wedge x_m \leq 10 \wedge t_m - x_m \leq 190) \end{aligned}$$

which can be shown to be a proper invariant of $modeler_2$. Table 4 shows the guards of the controllable actions of $\mathrm{TS}_2 = \|_{\Sigma}\{modeler_2, data, pulser\}$ and their successive restrictions.

The attempt to schedule the timed system $\mathrm{TS}'_2 = \mathrm{TS}_2/(K_{\mathrm{mutex}} \wedge K_{pr^{data\_exp}})$ as above by giving pulser priority over $modeler_2$ if data is fresh, fails. Timing analysis shows that $K^2_{\mathrm{sched}} = K^{modeler_2}_{\mathrm{sched}} \wedge K^{data}_{\mathrm{sched}} \wedge K^{pulser}_{\mathrm{sched}}$ is no more a proper invariant of $\mathrm{TS}'_2/K_{pr^{CPU}}$, since the priority of *pulser* over $modeler_2$ as long as *data* is *fresh* makes $modeler_2$ miss its

*Table 4*. Successive restrictions of the guards of controllable actions.

| | $\mathrm{TS}_2$ | $/(K_{\mathrm{mutex}} \wedge K_{pr^{data\_exp}})$ | $/K_{\mathrm{pr}_2^{\mathrm{CPU}}}$ |
|---|---|---|---|
| compute | $t_m \leq 165$ | $t_m \leq 165$ $\wedge \neg(reading \vee writing)$ | $t_m \leq 165$ $\wedge \neg(reading \vee writing)$ $\wedge (expired \vee t_m - t_p \geq 55)$ |
| update | $t_m \leq 190$ | $t_m \leq 190$ $\wedge \neg(reading \vee writing)$ | $t_m \leq 190$ $\wedge \neg(reading \vee writing)$ $\wedge (expired \vee t_m - t_p \geq 55)$ |
| expire | $y \geq 130$ | $y \geq 130 \wedge \neg reading$ | $y \geq 130 \wedge \neg reading$ |
| read | $t_p \leq 85$ | $t_p \leq 85 \wedge \neg(computing$ $\vee updating \vee expired$ $\vee y \geq 130)$ | $t_p \leq 85 \wedge \neg(computing$ $\vee updating \vee expired$ $\vee y \geq 130) \wedge t_m - t_p \leq 55$ |

deadline. Liveness of the whole system is re-established by scheduling both processes under edf if *data* is *fresh*. This policy is specified by the priority rule

$$\begin{aligned} pr_2^{CPU} = \{&(fresh \wedge t_m - t_p < 200 - 145, \\ &\{compute \prec read, update \prec read\}) \\ &(fresh \wedge t_m - t_p > 200 - 145, \\ &\{read \prec compute, read \prec update\})\} \end{aligned}$$

Timing analysis of the restricted system $\mathrm{TS}_2'/K_{\mathrm{pr}_2^{\mathrm{CPU}}}$ where $K_{\mathrm{pr}_2^{\mathrm{CPU}}}$ is the constraint corresponding to $pr_2^{CPU}$, shows that $K^2_{\mathrm{sched}}$ is not a proper invariant. However, the system is correctly scheduled from the initial state where both $modeler_2$ and *pulser* are *sleeping*, *data* is *expired*, and all timers are zero. In other words, there is a non-empty constraint implying $K^2_{\mathrm{sched}}$ that holds for this initial state, and that is a proper invariant of $\mathrm{TS}_2'/K_{pr_2^{CPU}}$. We have obtained a correctly scheduled system by successively refining scheduling policies.

## 7. Conclusion

The paper presents a methodology for scheduler modeling based on the controller synthesis paradigm. The methodology relies on the idea that the model of the scheduled system can be obtained by successive and appropriate restrictions of the guards of controllable actions of a model representing the real-time application. The resulting abstract scheduler specification is a function associating with system states sets of enabled controllable actions. In the proposed methodology, we privilege abstraction and minimality of concepts and we do not address concepts related to implementation description. Nevertheless, the proposed decomposition reflects a separation of issues that is more or less respected in practice, no matter how each layer is implemented.

This methodology is a framework for unifying existing scheduling theory and work on scheduler extraction from models of real-time applications. The decomposition of scheduling requirements into schedulability requirements, $K_{\mathrm{sched}}$, and policy requirements, $K_{\mathrm{pol}}$, allows better understanding the two approaches. Scheduling theory studies sufficient conditions guaranteeing $K_{\mathrm{sched}}$ for particular scheduling algorithms characterized by some $K_{\mathrm{pol}}$. Usually, modeling based approaches consist in extracting from a timed model schedulers which satisfy $K_{\mathrm{sched}}$ without taking into account particular scheduling policies. We recommend the application of policy constraints as a means to reduce the complexity of the modeling based approach.

The methodology is based on the use of results on control invariants and their composability. Control invariance and composability of policy requirements reduce the scheduling problem to the search for control invariants implying $K_{\mathrm{sched}}$. This is the hard problem for which tool support and/or user ingenuity are necessary. Consistency of restriction constraints and of policy requirements in particular, is crucial for the application of the approach; inconsistent constraints may introduce deadlocks that make the application non-schedulable. Some solutions to this problem have already been

studied in Bornot et al. (1997) where deadlock-freedom preservation results are given for both priority and mutual exclusion constraints.

## Acknowledgment

The authors thank Sergio Yovine for constructive remarks.

## References

Altisen, K., Gößler, G., Pnueli, A., Sifakis, J., Tripakis, S., and Yovine, S. 1999. A framework for scheduler synthesis. In *RTSS'99 Proceedings*, pp. 154–163.

Altisen, K., Gößler, G., and Sifakis, J. 2000. A methodology for the construction of scheduled systems. In M. Joseph (ed.), *Proc. FTRTFT 2000*, Vol. 1926 of *LNCS*, pp. 106–120.

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

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

Baeten, J., Bergstra, J., and Klop, J. 1986. Syntax and defining equations for an interrupt mechanism in process algebra. *Fundamenta Informaticae* IX(2): 127–168.

Ben-Abdallah, H., Choi, J., Clarke, D., Kim, Y., Lee, I., and Xie, H. 1998. A process algebraic approach to the schedulability analysis of real-time systems. *Real-Time Systems* 15(3): 189–219.

Ben-Abdallah, H., Lee, I., and Sokolsky, O. 1999. Specification and analysis of real-time systems with PARAGON. In *Annals of Software Engineering*.

Bertin, V., Poize, M., and Sifakis, J. 2000. Towards validated real-time software. In *Proc. 12th Euromicro Conference on Real Time Systems*, pp. 157–164.

Bornot, S., Gößler, G., and Sifakis, J. 2000. On the construction of live timed systems. In *Proc. TACAS 2000*, Vol. 1785 of *LNCS*.

Bornot, S., and Sifakis, J. 2000. An algebraic framework for urgency. *Information and Computation* 163: 172–202.

Braberman, V. 2000. Modeling and checking read-time system designs. Ph.D. thesis, Departamento de Computación, Universidad de Buenos Aires.

Cleaveland, R., Lüttgen, G., Natarajan, V., and Sims, S. 1996. Priorities for modeling and verifying distributed systems. In *Proc. TACAS'96*, pp. 278–297.

Daws, C., Olivero, A., Tripakis, S., and Yovine, S. 1996. The tool KRONOS. In *Hybrid Systems III, Verification and Control*, Vol. 1066 of *LNCS*, pp. 208–219.

Elseaidy, W., Cleaveland, R., and Baugh, J. Jr. 1997. Modeling and verifying active structural control systems. *Science of Computer Programming* 29(1–2): 99–122.

Ericsson, C., Wall, A., and Yi, W. 1999. Timed automata as task models for event-driven systems. In *Proc. RTCSA'99*.

Gößler, G. 2001. PROMETHEUS—A compositional modeling tool for real-time systems. In P. Pettersson and S. Yovine (eds), *Proc. Workshop RT-TOOLS 2001*.

Härbour, M., Klein, M., Obenza, R., Pollak, B., and Ralya, T. 1993. *A Practitioner's Handbook for Real-Time Analysis*. Kluwer.

Henzinger, T., Ho, P., and Wong-Toi, H. 1997. HYTECH: A model checker for hybrid systems. In *Software Tools for Technology Transfer*, pp. 110–122.

Henzinger, T. A., Horowitz, B., and Kirsch, C. M. 2001. Embedded control systems development with Giotto. In *Proc. LCTES 2001* (to appear).

Jensen, H., Larsen, K., and Skou, A. 2000. Scaling up UPPAAL: automatic verification of real-time systems using compositionality and abstraction. In *Proc. FTRTFT 2000*.

Kang, I., Lee, I., and Kim, Y. S. 1996. An efficient space generation for the analysis of real-time systems. In *Proceedings of the International Symposium on Software Testing and Analysis*.

Keshav, S. 1997. *An Engineering Approach to Computer Networking*. Addison-Wesley.

Kwak, H.-H., Lee, I., Philippou, A., Choi, J.-Y., and Sokolsky, O. 1998. Symbolic schedulability analysis of real-time systems. In *RTSS 1998 Proceedings*, pp. 409–418.

Lin, F., and Wonham, W. 1988. Decentralized supervisory control of discrete-event systems. *Information Sciences* 44: 199–224.

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

Maler, O., Pnueli, A., and Sifakis, J. 1995. On the synthesis of discrete controllers for timed systems. In E. Mayr and C. Puech (eds), *STACS'95*, Vol. 900 of *LNCS*, pp. 229–242.

Mok, A. 1983. Fundamental design problems for the hard real-time environments. Ph.D. thesis, MIT.

Niebert, P., and Yovine, S. 2000. Computing optimal operation schemes for chemical plants in multi-batch mode. In *Proceedings of Hybrid Systems, Computation and Control*, Vol. 1790 of *LNCS*.

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

Sha, L., Rajkumar, R., and Lehoczky, J. P. 1990. Priority inheritance protocols: an approach to real-time synchronization. *IEEE Transactions on Computers* 39(9): 1175–1185.

Vestal, S. 1994. Fixed-priority sensitivity analysis for linear compute time models. *IEEE Transactions on Software Engineering* 20(4): 308–317.