Available online at www.sciencedirect.com

Mathematics and Computers in Simulation 70 (2006) 408–418

www.elsevier.com/locate/matcom

Grid automata and supervisory control of dense real-time discrete event systems

Mustapha Nourelfath[a,*], Ahmed Khoumsi[b]

[a] Department of Mechanical Engineering, Network Organization Technology Research Center (CENTOR), Université Laval, Que., Canada
[b] Department of Electrical and Computer Engineering, University of Sherbrooke, Que., Canada

Available online 27 December 2005

Abstract

We present a generalization of the classical supervisory control theory for discrete event systems to a setting of dense real-time systems modeled by Alur and Dill timed automata. The main problem involved is that in general the state space of a timed automaton is (uncountably) infinite. The solution is to reduce the dense time transition system to an appropriate finite discrete subautomaton, the *grid automaton*, which contains enough information to deal with the timed supervisory control problem (TSCP). The plant and the specifications region graphs are sampled for a granularity defined in a way that each state has an outgoing transition labeled with the same time amount. We redefine the controllability concept in the context of grid automata, and we provide necessary and sufficient solvability conditions under which the optimal solution to centralized supervisory control problems in timed discrete event systems under full observation can be obtained. The enhanced setting admits subsystem composition and the concept of forcible event. A simple example illustrates how the new method can be used to solve the TSCP.
© 2005 IMACS. Published by Elsevier B.V. All rights reserved.

Keywords: Timed discrete event systems; Supervisory control; Automatic synthesis; Timed automata; Grid automata

1. Introduction

Discrete event systems (DES) are systems with a discrete state space and a state evolution which is determined by events. They are basically asynchronous (not clock driven) and evolve in accordance with the abrupt occurrence, at possibly unknown irregular intervals, of physical events. For example, an event may correspond to the completion of a task or the failure of a machine in a manufacturing system, the transmission of a packet in a communication system, the arrival or departure of a customer in a queue, or the occurrence of a disturbance or change of set-point in a complex control system. Such systems are encountered in a variety of fields, for example, in robotics, computer and communication networks, traffic, logistics and manufacturing. In the last decade, the complex interaction of highly flexible process equipment with computer-based control systems has stressed the need for sound analysis, design, validation and implementation methods for DES. Different techniques and methods may be exploited, such as knowledge engineering [15,9,5], Petri nets [18,13], min–max algebra [15,5], perturbation analysis [16,8], and the supervisory control theory (SCT) [23,27]. The work presented here is based on the SCT.

Ramadge and Wonham in their work [23,27] on supervisory control of DES have developed a theory for the synthesis of a controller that ensures the closed loop system has a desirable behavior. In the framework of automata and formal

[*] Corresponding author. Tel.: +1 418 656 2131x12355; fax: +1 418 656 7415.
E-mail address: Mustapha.Nourelfath@uqat.ca (M. Nourelfath).

0378-4754/$32.00 © 2005 IMACS. Published by Elsevier B.V. All rights reserved.
doi:10.1016/j.matcom.2005.11.009

languages, the SCT (also called Ramadge and Wonham or RW theory) has successfully treated a variety of abstract synchronization problems defined by specifications of a qualitative or "logic-based" type, abstracting away the actual timing delays between events. Most research in this field has concentrated on the logical sequencing of events. Real-time DES are systems for which, in addition to the correct order of events, constraints on delays separating certain events must be satisfied. Such systems occur in many safety–critical applications. Timing introduces a new dimension of DES modeling and control, of considerable applied interest but also of significant complexity. To develop the supervisory control of timed discrete event systems (TDES), two models have been considered: *discrete time* model [7,3,22] and *continuous or dense time* model [2].

In a discrete time model, the domain of integers is used to describe time. This model specifies a priori the smallest measurable time unit [21,6,17]. A conceptual global digital clock is used to generate a fictitious event *tick* at a constant frequency. The discrete time model can be appropriate for certain kinds of synchronous digital circuits, where signal changes are considered to have occurred exactly when a clock signal arrives. It is conceptually simple to manipulate discrete time models using finite automata, but the compensating disadvantage is that time is represented only in an approximate sense.

In the continuous time model of Alur and Dill [2,10], time measures are modeled by real variables and represent exact values of time. The association of an exact occurrence time with each event imposes a minimal set of restrictions on the modeling framework. Events may occur arbitrarily close to one another and their timing information is modeled exactly. The dense time model is more expressive than the discrete time model. Furthermore, such a model is needed for correctness. In [1], the author gives an example of an asynchronous circuit subject to bounded inertial delays, where fixing in advance the time quantum for the discrete time and fictitious clocks models gives an incorrect reachability analysis. Therefore, a dense model of time is appropriate. It is also more natural for physical processes operating over continuous time. But, dealing with dense time in a finite automata framework requires the transformation of a set of dense time traces into an ordinary formal language.

The purpose of this paper is to provide a method for the supervisory control of real-time DES modeled by Alur and Dill's timed automata. The main problem involved is that in general the state space of a timed automaton is uncountably infinite. Alur and Dill in [2] proved that even the task of analyzing whether a timed automaton accepts any timed trace at all is computationally difficult. Wong-Toi and Hoffmann proved also that the supervisory control problem of systems modeled by timed automata of Alur and Dill is a hard problem [26]. This difficulty is because of the infinity of delay transitions. To overcome this problem, a discretization of the state space is required which is still sufficiently refined to avoid any possible illegal behavior in the system closed loop behavior. All the works within the RW theory on dense timed automata, that we are aware of, are based on transforming timed automata into region graphs [26,14,20,4]. In [26,14], the authors solve the control problem by completely discretizing the timed automata into region graphs and then solve the discrete synthesis problem. The method of [20], by working directly on the timed automaton, makes only the discretization necessary to solve the control problem. In [4], the authors demonstrate how the synthesis problem for real-time systems can be formulated and solved symbolically. Our supervisory control method for dense real-time systems is based on the concepts of *sampling* and *grid automata*. We present a sampling algorithm to construct grid automata of the plant and the specifications. The idea behind the construction of the grid automaton is to represent each clock region with a finite set of clock interpretations, referred to as *representatives* of the clock region. In a grid automaton each clock region is represented at least by one representative. A grid automaton is a finite subautomaton of the state space constructed on the basis of an appropriate discretization of the timed automaton. For the supervisory control problem, we determine the smallest time difference that we need to consider between two states that differ only in the value of the clock variables. This time difference is called the *sampling granularity*. The plant and the specifications region graphs are sampled for a granularity defined in a way that each state has an outgoing transition labeled with the *same* time amount. It follows that our approach has the advantage, over the existing approaches, to allow the synchronization between the plant and the specifications and synchronous product operations of their automata models. Furthermore, this time amount can be treated as a discrete "untimed" transition. Therefore, suitable adaptations of the classical RW theory can be applied to the grid automata of the plant and the specifications. We provide necessary and sufficient solvability conditions under which the optimal solution to centralized supervisory control problems in TDES under full observation can be obtained. In our approach, the supervisor can actually force events; while in [26,20,4], the authors make the assumption that a supervisor can only enable or disable events in analogy to the untimed model rather than force them upon the plant. This is a strong restriction because in most systems the supervisor can actually force or schedule events, just like the plant, since forcible events arise naturally in the presence of timing. However, the distinction of forcible events is only necessary as a result of the abstraction from timed to discrete automata, since any event in timed automata can be forced if appropriate guards and invariants are specified by a timed automaton. The semantics of our dense time supervisory control model borrows ideas from the discrete time supervisory control model of Brandin and Wonham [6] to accommodate scheduling capabilities of the supervisor.

The remainder of this paper is as follows: Section 2 presents the model of timed automata used to describe the plant and the specifications and its operational semantics. Our supervisory control method for TDES is developed in Section 3, and illustrated by a simple example in section 4. Conclusions are drawn in Section 5.

2. A dense time model

2.1. Notations

Let $\mathbf{R}$ denote the set of reals, $\mathbf{R}^{\geq 0}$ the set of nonnegative reals, and $\mathbf{R}^{\infty}$ the set of reals together with the single element ∞. We use ∞ and $+\infty$ as synonyms. $\mathbf{N}$ is the set of nonnegative integers and $\mathbf{N}^{\infty} = \mathbf{N} \cup \{\infty\}$. For any $t \in \mathbf{R}$, $\lfloor t \rfloor$ denotes the integer part of t, and fract(t) denotes the fractional part of t; that is, $t = \lfloor t \rfloor + \text{fract}(t)$. The term "derivative" means "derivative with respect to physical time", and "reset" means "set to zero". Given a set E, the notation 2^E denotes as usually the set of its subsets.

2.2. The model

Continuous time can be viewed as a real variable that evolves indefinitely and whose derivative is equal to 1. A clock is a continuous time whose value can be reset at the occurrence of an event. To express system behaviours with timing constraints, we consider finite graphs augmented with a finite set of clocks. The vertices of the graph are called locations, and edges are called transitions. While transitions are instantaneous, time can elapse in a location. With each transition we associate a clock constraint, and require that the transition may be taken only if the current values of the clocks satisfy this constraint. With each location we associate a clock constraint called its invariant, and require that time can elapse in a location only as long as its invariant stays true.

To define timed automata formally, we need to say what type of clocks constraints are allowed as invariants and enabling conditions. An atomic constraint compares a clock value with a time constant, and a clock constraint is a conjunction of atomic constraints. Any value from $\mathbf{Q}$, the set of nonnegative rationals, can be used as a time constant. An enabling condition is a clock constraint or guard. We consider a set of clock variables C, and we define an enabling condition as being any formula or conjunction of formulas of the form "c op k", where op $\in \{<, >, \leq, \geq, =\}$, $c \in C$ and $k \in \mathbf{N}$. Let $\Phi(C)$ be the set of enabling conditions. The choice of natural as bounds in constraints will help us, later, in the discretization of the set of reals into integers, reducing thereby the state space of timed systems. A timed automaton (TA) A is a tuple (Σ, L, l_0, C, T) where:

- Σ is a finite set of events.
- L is a finite set of locations.
- l_0 is the initial location.
- C is a finite set of clocks. Each clock $x \in C$ has a domain $[0, C_x] \cup \{\infty\}$, where C_x is the largest integer constraint appearing in a constraint over clock x.
- $T \subseteq L \times L \times \Sigma \times 2^C \times \Phi(C)$ is the set of transitions.

A tuple $(l, l', a, \lambda, \Delta) \in T$, denoted by $l \xrightarrow{a,\lambda,\Delta} l'$, represents a transition from location l to location l' on event a. Δ is a clock constraint over C that specifies when the transition is enabled, and the set $\lambda \subseteq C$ gives the clocks to be reset with this transition.

In this definition, each clock x is relevant only under the integer constant C_x. We will represent each clock value greater than this constant by ∞, and we write:

$$\forall \varepsilon > 0, \quad \forall x \in C, \quad C_x + \varepsilon = \infty.$$

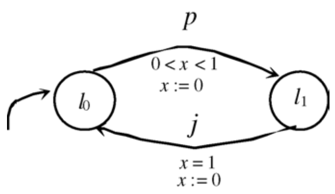

Fig. 1. A timed automaton example.

Consider the timed automaton of Fig. 1. The initial location is l_0. There is a single clock x. The constraint $(0 < x < 1)$ specifies that the transition from l_0 to l_1 is enabled when the clock value is between 0 and 1. When the system switches to location l_1 on event p, the clock x gets reset to 0. While in location l_1, the value of the clock x shows the time elapsed since the occurrence of the last transition. The transition from l_1 to l_0 on event j is enabled only if this value is equal to 1 (constraint $x = 1$). When the system switches to location l_0 (on event j), the clock x gets reset to 0 again.

2.3. Clock region

A clock interpretation or valuation ν for a set C of clocks assigns a real value to each clock; that is, it is a mapping from C to the set $\mathbf{R}^{\infty}$. The set of clock valuations is denoted by $V(C)$. We say that a clock interpretation ν for C satisfies a clock guard or constraint Δ, denoted $\nu| = \Delta$, if and only if Δ holds under ν (i.e., Δ evaluates to true according to the values given by ν). For $d \in \mathbf{R}^{\geq 0}$, $\nu + d$ denotes the clock interpretation which assigns a value $\nu(x) + d$ to each clock x. For $X \subseteq C$, $[X := d]\nu$ denotes the clock interpretation for C which assigns the value d to each $x \in X$, and agrees with ν over the rest of the clocks.

The operational semantics of a timed automaton A is defined by associating with it a timed labeled transition system $\mathrm{TS}_A = (S, \Sigma \cup \mathbf{R}^{\geq 0}, \rightarrow, (l_0, \nu_0))$. A state s of TS_A $(s \in S)$ is a pair (l, ν) where l is location $(l \in L)$ and ν is a clock interpretation. The pair (l_0, ν_0), where $\nu_0(x) = 0$ for each clock $x \in C$, represents the initial state of A. $\Sigma \cup \mathbf{R}^{\geq 0}$ is the label-set (both Σ and time increments), and the transition relation $(\rightarrow)$ is defined for any element in $\Sigma \cup \mathbf{R}^{\geq 0}$. There are two types of transitions in TS_A:

- State can change due to elapse of time. The transitions of this type have the time additivity property: $s \xrightarrow{\varepsilon(d+d')} s' \Leftrightarrow \exists s'' \in S : s \xrightarrow{\varepsilon d} s'' \wedge s'' \xrightarrow{\varepsilon d} s'$ (ε denotes the empty event).
- State can change due to a location-switch: for a state (l, ν) many delays transitions are executed and when the automation is in a state (l, ν') where the temporal constraint of an outgoing transition is satisfied, the automaton can execute this transition. Such explicit transitions are denoted by $s \xrightarrow{a} s'$, with $a \in \Sigma$.

The timed labeled transition system TS_A is infinite, because of the infinity of delay transitions. To reduce the system state space, we use an equivalence relation [3] on the set of clock valuations $V(C)$ in order to cluster equivalent states of TS_A into equivalent classes.

The equivalence relation $\sim$, called *region equivalence*, is defined over the set of all clock interpretations for C. For two interpretations ν and ν', $\nu \sim \nu'$, if and only if all the following conditions hold:

- $\forall x \in C$, either $\lfloor \nu(x) \rfloor = \lfloor \nu'(x) \rfloor$, or $\nu(x) > C_x$ and $\nu'(x) > C_x$.
- $\forall x, y \in C, ((\nu(x) \leq C_x) \wedge (\nu(y) \leq C_y))$.
- $(\mathrm{fract}(\nu(x)) \leq \mathrm{fract}(\nu(y))) \Leftrightarrow (\mathrm{fract}(\nu'(x)) \leq \mathrm{fract}(\nu'(y)))$.
- $\forall x \in C \mid (\nu(x) \leq C_x)$, $(\mathrm{fract}(\nu(x)) = 0 \Leftrightarrow \mathrm{fract}(\nu'(x)) = 0)$.

A *clock region* for A is an equivalence class of clock valuations induced by $\sim$. Let $[\nu]$ denotes the clock region to which ν belongs. Note that there are only a finite number of regions. An upper bound of this number has been given in [3] and optimized in [11].

The clock regions of the TA of Fig. 1 are represented in Fig. 2. We distinguish here between two kinds of clock regions: the corner points $\{R_1, R_2\}$, and the open line segments $\{R_3, R_4\}$.

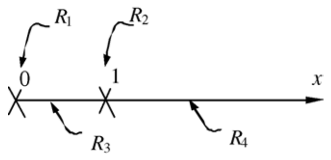

Fig. 2. Clock regions.

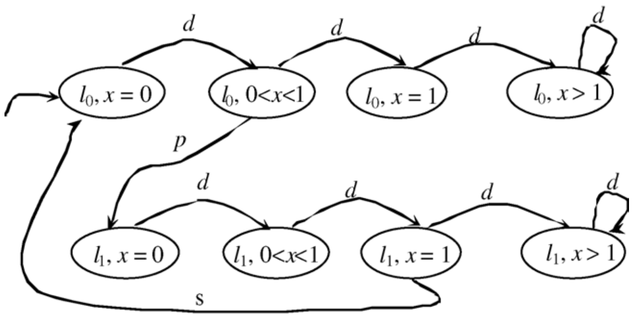

Fig. 3. Region automaton.

2.4. Region automaton

Region equivalence relation $\sim$ over clock interpretations is extended to an equivalence relation over the state space by requiring equivalent states to have identical locations and region-equivalent clock interpretations: $(l, \nu) \sim (l', \nu')$ if and only if $l = l'$ and $\nu \sim \nu'$. The quotient of a timed automaton with respect to the region equivalence is called the *region automaton* of A, and is denoted R(A). The region automaton $R(A)$ of a timed automaton $A = (\Sigma, L, l_0, C, T)$ is quadruple $(\Sigma \cup \mathbf{R}^{>0}, S, s_0, T)$, such that:

- $S = \{(l, [\nu]) \mid l \in L \wedge \nu \in V(C)\}$.
- $s_0 = \langle l_0, [\nu_0] \rangle$, with $\nu_0(x) = 0$ for all $x \in C$.
- $R(A)$ has a transition $s \xrightarrow{a} s'$ from $s = (l, [\nu])$ to $s' = (l', [\nu'])$ at the occurrence of the event $a \in \Sigma$ if and only if A has a transition $l \xrightarrow{a,\lambda,\Delta} l'$ such that $\nu| = \Delta$ and $\nu' = [\lambda := 0]\nu$.
- $R(A)$ has a delay transition $s \xrightarrow{d} s'$ from $s = (l, [\nu])$ to $s' = (l', [\nu'])$ on time increment $d \in \mathbf{R}^{>0}$, if and only if $[\nu'] = [\nu + d]$.

Thus, the region automaton is a partition of the uncountable state space of the timed automaton into a finite number of clock regions, a set of equivalent states in other words. The region automaton is at the heart of any verification, testing or synthesis technique for timed systems. This automaton corresponds in a certain manner to the reachability analysis graph of the timed system. Our definition of region automaton is slightly different from the definition in [3]. We take into account the delay transitions. Such a definition is similar to the one given in [12] for timed input output automata. It leads to automata like those called *d-regions* in [14]. Also, the region automaton obtained by this definition accepts the same timed traces as in [26]. Note that the event d does not represent the same amount of time. The corresponding region automaton for Fig. 1 TA is shown in Fig. 3.

3. Supervisory control synthesis for dense real-time systems

In this section, we introduce the notion of conformance relation, and we show how to convert a timed automaton into a minimal grid automaton, retaining sufficient information for analyzing the closed loop system. The controllability is defined in the context of grid automata, and a procedure is suggested for finding the maximally permissive supervisor in a timed supervisory control problem.

3.1. Specification, implementation, and conformance relation

In general, a design problem can be defined as: given a specification, find an implementation that satisfies the specification. A design problem can be considered as a supervisory control problem if the implementation consists of an already existing uncontrolled process G and a still-to-be-designed supervisor S. In this paper, we analyze the control problem of finding a timed supervisor S such that the synchronous composition of S and G, i.e., the implementation $(G||S)$ is *conform* to a specification E. The model we use to represent G and E is the timed automata of Alur and Dill [2] as defined in Section 2, and which consists of all clocks having bounded domains [24] to indicate that clock values are relevant only under a certain integer constant. The starting point of our supervisory control of dense real-time DES is the definition of the conformance relation between an implementation and the specification. This conformance relation defines the meaning of "an implementation is conform to its specification". Many conformance relations have been introduced and used to synthesize supervisors from logical automata models. However, like the pioneer work of Ramadge and Wonham [23], the majority of the existing works on RW theory use the trace-equivalence (usually in an implicit way). In the context of dense real-time systems, we define in this paper the conformance relation between an implementation and its specification as a trace-equivalence. The trace-equivalence relation is based on the notion of traces of a state. By definition, a sequence of events σ is a trace of a state s if the automaton can evolve from s to another state s' on σ (s' can be equal to s). We denote the set of traces of a state s by traces(s) and we say that two states s and s' are trace-equivalent, notation $s =_{\text{trace}} s'$, if and only if traces(s) = traces(s'). Furthermore, two automata S and I with their respective initial states s_0 and i_0, are trace-equivalent, notation $S =_{\text{trace}} I$, if and only if $s_0 =_{\text{trace}} i_0$. The trace-equivalence is an equivalence relation.

3.2. Sampling concept and grid automata

To ensure the trace-equivalence between the implementation $(G||S)$ and the specification (E), our approach for supervisory control synthesis is based on the sampling of the region graph. By using a known granularity, we make explicit the elapsing of time and we obtain a reduction called the grid automaton. The concept of sampling has been introduced in [19] for the verification of real-time systems, and also used in [25,12] for timed test case generation. In this paper, we use sampling to synthesize supervisors for real-time DES.

The idea behind the construction of the grid automaton is to represent each clock region with a finite set of clock interpretations, referred to as representatives of the clock region. Thus, in a grid automaton each clock region is represented at least by one representative. The grid automaton has the property that each state has an outgoing delay transition labeled with the granularity of sampling. This granularity depends on the number of clocks used in TA. To determine the set of representatives of each clock region, grid points with granularity $1/k$ are defined in [19], where it is proven that for each clock region of a TA with n clocks, there will exist a set of its representatives in the grid points with a granularity of at most $1/(n+1)$. However, to represent all the regions reachable by delay transitions, we have to use for $n \geq 2$ a granularity of at most $1/(n+2)$ [12]. For the supervisory control problem, we use such a granularity; the construction of grid automata leads to the explicit extension of the alphabet with an event denoted by gran which is $1/(n+2)$ if $n \geq 2$ and $1/2$ if $n = 1$.

An algorithm for grid automata construction can be found in [12]. This algorithm can be summarised as follows. Given an n-clock TA, we first derive the maximal granularity (if $n = 1$ then the granularity is $1/2$ else the granularity is $1/(n+2)$). In a second step, we create the initial state formed with the initial location of the TA and a valuation that sets all clocks to zero. In a third step, we create all states reachable from the initial state with gran delay transitions. Then for each state (l, ν), create a transition $((l, \nu), a, (l', [\lambda := 0]\nu))$ for each transition $(l, l', a, \lambda, \Delta)$ in the TA such that ν satisfies Δ. Afterwards, we repeat the same process starting with state $(l', [\lambda := 0]\nu)$. The third step is repeated until all the states are treated.

Let us emphasise that in our supervisory control problem, the value n we have to consider for the granularity computation constitutes the total number of clocks used in both the process and the specifications TA models.

The obtained grid automata are minimized. Also, their synchronous compositions can be performed. Note that in our approach such a *synchronization* of the grid automata is possible because of the use of the *same* delay transition.

3.3. Supervisory control synthesis

3.3.1. Supervisory control over grid automata

Let $G = (Q, \Sigma_e, \delta, Q_m, q_0)$ be the labeled transition system representing the grid automaton of the plant. The alphabet Σ_e is given by: $\Sigma_e := \Sigma \cup \{\text{gran}\}$, where Σ is the initial alphabet of the plant TA and gran is the sampling granularity (representing the progress of time).

To use grid automata as models for supervisory control, it is necessary to specify the ways in which the transitions can be controlled by an external agent or supervisor. Let $\Sigma_c \subseteq \Sigma_e$ be the set of *controllable* events (i.e. those events whose occurrence is either preventable or allowable), and let $\Sigma_{uc} \subseteq \Sigma_e$ be the set of *uncontrollable* events (i.e. those events which cannot be prevented). Furthermore, we consider another category of events that arises naturally in the presence of timing: the *forcible* events or elements of a new subset $\Sigma_{for} \subseteq \Sigma$. By analogy with [6], a forcible event is one that can preempt an event gran; in this case, the event gran can be treated as a controllable event though it is normally treated as an uncontrollable event. Note that a forcible event may be controllable or uncontrollable. A forcible event that is uncontrollable cannot be directly prevented from occurring by disablement. Also, while formally designated controllable to simplify terminology, the status of gran lies intuitively between controllable and uncontrollable: no technology could prohibit gran in the sense of stopping the clock, although a forcible event, if eligible may preempt it.

3.3.2. The timed supervisory control problem

Before introducing the discrete time supervisory control problem, let us define the subset of eligible events, say $\text{Elig}_G(s) \subseteq \Sigma_e^*$, as follows $\text{Elig}_G(s) := \{\sigma \in \Sigma_e | s\sigma \in L(G)\}$. Here, the language $L(G)$ is, of course, defined over the full alphabet Σ_e including gran. Henceforth, we shall use the term "eligible" in this extended sense to apply to gran as well as to events in the timed automata alphabet Σ. The objective here is to synthesize a supervisor whose task is to enable, disable and force events in the process such that the resulting supervised behavior obeys some temporal and/or logical specifications.

Formally, a supervisor S is defined to be any map $\overline{h} : L(G) \to \psi(\Sigma)$ such that, for all $s \in L(G)$:

$$\overline{h}(s) \cap \text{Elig}_G(s) \neq \phi \quad \text{and} \quad \overline{h}(s) \supseteq \begin{cases} \Sigma_{unc} \cup \{\text{gran}\}, & \text{if } \overline{h}(s) \cap \Sigma_{for} \text{ is empty} \\ \Sigma_{unc}, & \text{if } \overline{h}(s) \cap \Sigma_{for} \text{ is not empty} \end{cases}.$$

Intuitively, $\overline{h}(s)$ is the set of events (including gran) that are enabled (by the supervisor) after the execution of the sequence s. Note that the supervisor forces events of Σ by disabling gran (i.e., by preempting time).

As usual, the closed behavior of $\overline{h}/G$ is defined by the language $L(\overline{h}/G) \subseteq L(G)$. A supervisor $\overline{h}$ is said to be nonblocking for G if $L(\overline{h}/G) = \overline{L_m(\overline{h}/G)}$.

The timed supervisory control problem (TSCP) considered can be formulated as follows:

Given *a grid automaton G* corresponding to a plant timed automaton modeled over an alphabet Σ (with controllable events Σ_c and forcible events Σ_{for}) and given a *specification language* $K \subseteq L(G)$ (K is recognized by a grid automaton) find *a nonblocking supervisor S* such that $L(S/G) \subseteq K$.

3.3.3. Solution of the TSCP

To describe a solution of the above problem, the *controllability* concept is reconsidered in order to take into account the forcibility of events. Let $K \subseteq L(G)$ be arbitrary, and write $\text{Elig}_K(s) := \left\{\sigma \in \Sigma_e | s\sigma \in \overline{K}\right\}, \quad s \in \Sigma_e^*$.

The language K is defined to be controllable (with respect to G) if, for all $s \in K$:

$$\text{Elig}_K(s) \supseteq \begin{cases} \text{Elig}_G(s) \cap (\Sigma_{unc} \cup \{\text{gran}\}), & \text{if } (\text{Elig}_K(s) \cap \Sigma_{for}) \text{ is empty} \\ \text{Elig}_G(s) \cap \Sigma_{unc}, & \text{if } (\text{Elig}_K(s) \cap \Sigma_{for}) \text{ is not empty} \end{cases}.$$

The controllability of a language K with respect to a plant G means that an event σ (in the full alphabet including gran) may occur in K if:

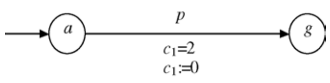

Fig. 4. The bus model.

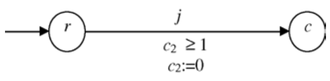

Fig. 5. The pedestrian model.

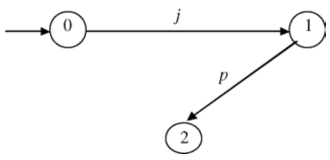

Fig. 6. The specification model.

(1) either it is uncontrollable or gran and currently eligible in the plant, and no forcible event is currently eligible in K and
(2) or it is definitely uncontrollable and eligible in G, but some forcible event is eligible in K.

The effect of the definition is to allow the occurrence of gran (when it is eligible in G) to be ruled out of K only when a forcible event is eligible in K and could thus be relied on to preempt it.

A supervisor may enable, disable or force events at any time during its observation of a sequence of events generated by the plant G. It allows only a subset of $L(G)$ to be generated. The synthesized supervisor will act on events such that the resulting supervised behavior obeys the specifications. If we think of K as a set of desirable *timed sequences*, then we want to know when it will be impossible to stop an *illegal sequence* from happening. If K is not controllable, a sub-language of K can be computed by procedures no different from procedures already described in the literature (e.g., [23,6]).

4. Example

We will illustrate our control with the bus/pedestrian example also used in [6,14]. The two TA of Figs. 4 and 5 model the bus and the pedestrian, respectively. The bus can make a single transition “pass” between the locations “approaching” and “gone”. The event *pass* occurs exactly two units of time after the instant when the bus starts *approaching*. The pedestrian can make a single transition “jump” between the locations “road” and “curb”. At least one unit of time elapses between the instant when the pedestrian decides to jump and the instant when he actually jumps. Two scenarios are possible, in which the pedestrian decides to jump as soon as the bus starts approaching. In the first scenario, the pedestrian jumps on the curb before the passage of the bus: sequence *jump pass*. In the second (undesirable) scenario, the bus passes and then the (run down) pedestrian jumps on the curb: sequence *pass jump*. The specification requires that the pedestrian jumps before the passage of the bus. The specification model is represented in Fig. 6. We consider that the event *jump* is controllable and forcible while the event *pass* is uncontrollable and unforcible.

Since two clocks are used in the TA of the plant and the specification, the sampling granularity is gran = 1/4 ($n \geq 2$ and gran = $1/(n+2)$). The following (Figs. 7–12) illustrate how the supervisor is constructed by means of the synthesis procedure developed in this paper. In these figures, the events *pass* and *jump* are represented by p and j, while the locations “approaching”, “gone”, “road” and “curb” are denoted by a, g, r and c (respectively). The obtained grid automaton of the supervisor leads by construction to the maximally permissive controllable behaviour of the closed loop TDES, when p is uncontrollable and j is forcible.

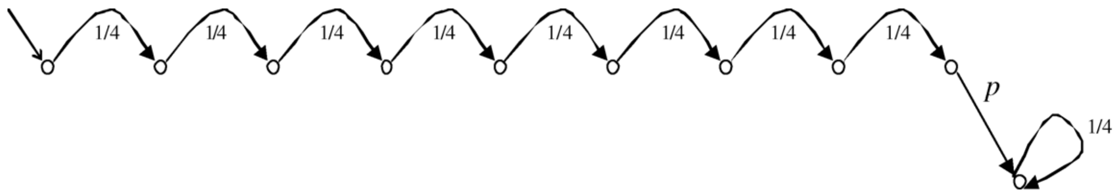

Fig. 7. The minimal grid automaton of the bus (*B*).

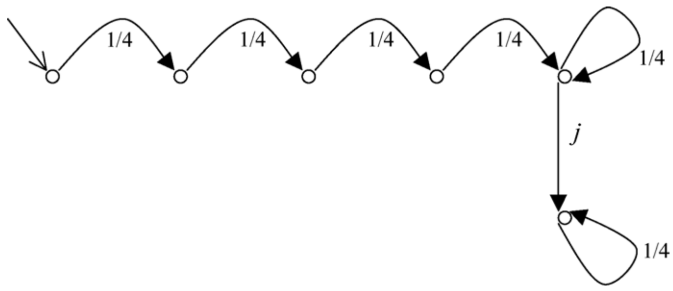

Fig. 8. The minimal grid automaton of the pedestrian (*P*).

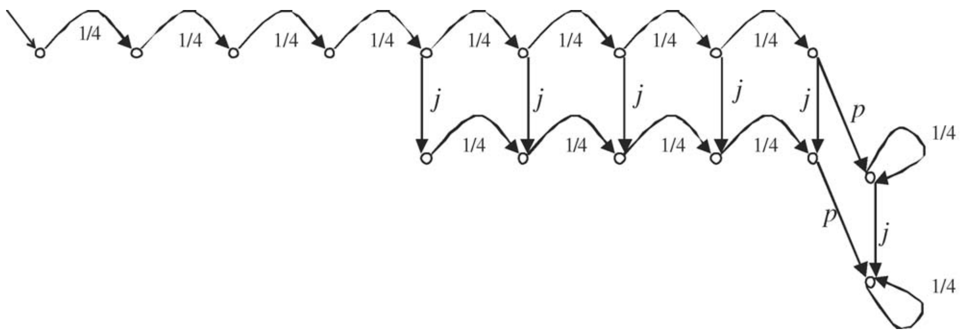

Fig. 9. The minimal grid automaton of the plant (*G*) obtained by the composition of *B* and *P*.

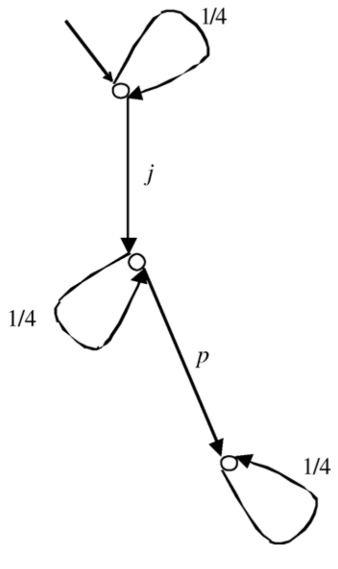

Fig. 10. The minimal specification grid automaton.

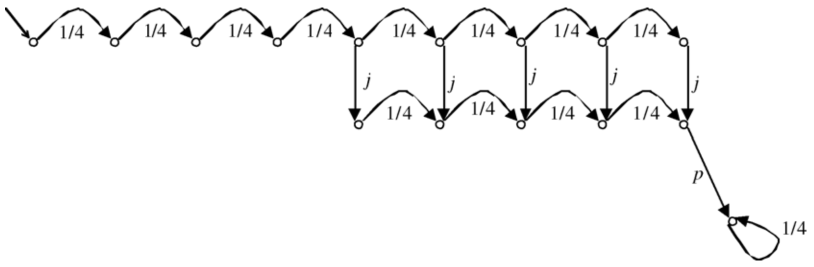

Fig. 11. The synchronous product of the plant and the specification.

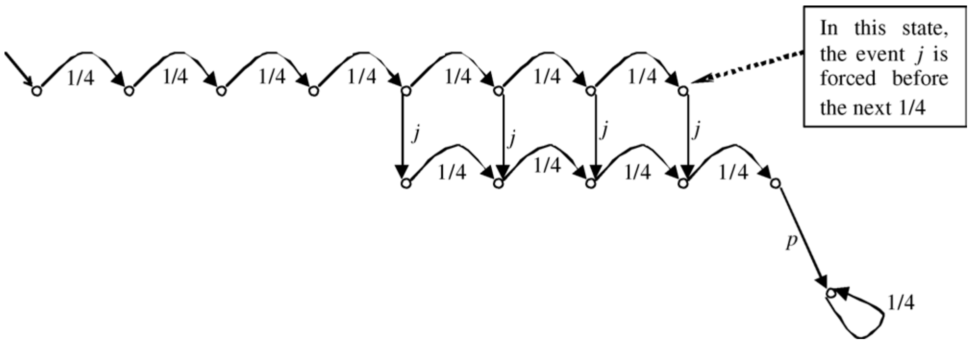

Fig. 12. The grid automaton of the supervisor.

5. Conclusions

In this paper, the supervisory control synthesis problem over dense real-time is solved by combining techniques developed mainly in [2,23,6,12]. The proposed timed supervisory control method is based on sampling region graphs of the process and the specifications. We show how to convert timed automata into minimal grid automata, retaining sufficient information for analyzing the closed loop system. The controllability concept is redefined in the context of grid automata, and a procedure is suggested for finding the maximally permissive supervisor in a timed supervisory control problem. Our approach admits both forcing and disablement as means of control. We assumed that the supervisor can precisely observe the whole configuration of the plant, including the values of all the relevant clocks. An interesting and important problem to deal with in future works is that of partial observation.

Acknowledgments

The authors would like to thank the Natural Sciences and Engineering Research Council of Canada (NSERC). The first thanks also the “Fonds de Recherche sur la Nature et la Technologie” of Quebec (NATEQ) and the FUQAT and FIR at the University of Quebec in Abitibi-Temiscamingue.

References

[1] R. Alur, Techniques for Automatic Verification of Real-Time Systems, Technical Report No. STAN-CS91-1378, Department of Computer Science, Stanford University, CA, Ph.D. Thesis, 1991.
[2] R. Alur, D. Dill, Automata for modeling real-time systems, in: Proceedings of the 17th Int. Coll. on Automata, Languages and Programming, Lecture Notes in Computer Science, vol. 443, Springer-Verlag, Warwick, UK, 1990.
[3] R. Alur, D. Dill, A theory of timed automata, Theor. Comp. Sci. 126 (1994) 183–235.
[4] E. Asarin, O. Maler, A. Pnueli, Symbolic controller synthesis for discrete and timed systems, in: Hybrid Systems II, Lecture Notes in Computer Science, vol. 999, Springer-Verlag, 1995, pp. 1–20.
[5] J.L. Boimond, J.L. Ferrier, Internal model control and max-algebra: controller design, IEEE Trans. Autom. Cont. 41 (3) (1996) 457–461.
[6] B. Brandin, W.M. Wonham, Supervisory control of timed discrete event systems, IEEE Trans. Autom. Cont. 39 (2) (1994) 329–351.
[7] Y. Brave, M. Heymann, Formulation and control of real time discrete event processes, in: Proceedings of 27th IEEE Conference on Decision and Control, New York, 1988, pp. 1131–1132.

[8] C.G. Cassandras, S. Lafortune, G.J. Olsder, in: Isidori Alberto (Ed.), Introduction to the Modelling, Control and Optimization of Discrete Event Systems, Trends in Control, A European Perspective, Springer-Verlag, Berlin, 1995, pp. 217–291.

[9] G. Cohen, D. Dubois, J.P. Quadrat, M. Viot, A Linear system theoretic view of discrete-event processes and its use for performance evaluation in manufacturing, IEEE Trans. Autom. Cont. 30 (1985) 210–220.

[10] D. Dill, Timing assumptions and verification of finite-state concurrent systems, in: Lecture Notes in Computer Sciences, vol. 407, Automatic Verification Methods for Finite State Systems, International Workshop, Springer-Verlag, Grenoble, France, 1989, pp. 197–212.

[11] A. Elqortobi, A. En-Nouaary, G.V. Bochmann, Dénombrement du nombre des régions dans un automate temporisé, Technical Report TR-1116, IRO Department, Montreal University, January 1998.

[12] A. En-Nouaary, R. Dssouli, F. Khendek, Timed Wp-method: testing real-time systems, IEEE Trans. Soft. Eng. 28 (11) (2002) 1023–1038.

[13] A. Giua, F. Dicesare, Petri net structural analysis for supervisory control, IEEE Trans. Robot. Autom. 10 (2) (1994) 185–195.

[14] A. Gouin, J.L. Ferrier, Supervisory control of timed automata, in: Proceedings of the European Control Conference (ECC), Karsruhe, Germany, 1999.

[15] B. Hayes-Roth, A blackboard architecture for control, Artif. Intell. 26 (3) (1985) 251–321.

[16] Y.C. Ho, Parameter sensivity of a statistical experiment, IEEE Trans. Autom. Cont. 24 (1979) 982–983.

[17] T.-J. Ho, A new approach to synthesis problems in timed discrete-event systems, Int. J. Cont. 73 (6) (2000) 505–519.

[18] B.H. Krogh, J. Magott, L.E. Holloway, On the complexity of forbidden state problems for controlled marked graphs, in: Poceedings of 30th IEEE Conference on Decision and Control, Brighton, England, 1991, pp. 85–91.

[19] K. Larsen, W. Yi, Time abstracted bisimulation: implicit specifications and decidability, in: Proceedings of Mathematical Foundations of Programming Semantics (MFPS), vol. 802 of Lecture Notes in Computer Science, Springer-Verlag, New York, 1993.

[20] O. Maler, A. Pnueli, J. Sifakis, On the synthesis of discrete controllers for timed systems, in: Proceedings of the 12th Annual Symposium on Theoretical Aspects of Computer Science, vol. 900, Springer-Verlag, 1995, pp. 229–242.

[21] J.S. Ostroff, Synthesis of controllers for real-time discrete event systems, in: Proceedings of the 28th Conference on Decision and Control, Tampa, FL, 1990, pp. 138–144.

[22] J.S. Ostroff, W.M. Wonham, A framework for real-time discrete event control, IEEE Trans. Autom. Cont. 35 (4) (1990) 386–397.

[23] J.G. Ramadge, W.M. Wonham, Supervisory control of a class of discrete event processes, SIAM J. Cont. Optim. 25 (1987) 206–230.

[24] J. Springetveld, F. Vaadranger, Minimizable timed automata, in: B. Jonsson, J. Parrow (Eds.), Proceedings of the 4th International School and Symposium on Formal Techniques in Real Time and Fault Tolerant Systems, vol. 1135 of Lecture Notes in Computer Sciences, Springer-Verlag, Uppsala, Sweden, 1996.

[25] J. Springetveld, F. Vaandrager, P. D'Argenio, Testing timed automata, Theor. Comp. Sci. 254 (2001) 225–257.

[26] H. Wong-Toi, G. Hoffmann, The Control of Dense Real-Time Discrete Event Systems, Technical Report STA-CS-92-1411, Stanford University, 1995.

[27] W.M. Wonham, J.G. Ramadge, On the supermal controllable sublanguage of a given language, Siam J. Cont. Optim. 25 (3) (1987) 637–659.