Timed vs. Time-Triggered Automata

Pavel Krčál[1], Leonid Mokrushin[1], P.S. Thiagarajan[2], and Wang Yi[1]

[1] Dept. of Information Technology, Uppsala University, Sweden
{pavelk,leom,yi}@it.uu.se
[2] Dept. of Computer Science, National University of Singapore
thiagu@comp.nus.edu.sg

Abstract. To establish a semantic foundation for the synthesis of executable programs from timed models, we study in what sense the timed language (i.e. sequences of events with real-valued time-stamps) of a timed automaton is recognized by a digital machine. Based on the non-instant observability of events, we propose an alternative semantics for timed automata. We show that the new semantics gives rise to a natural notion of digitalization for timed languages. As a model for digital machines we use time-triggered automata – a subclass of timed automata with simplified syntax accepting digitalized timed languages. A time-triggered automaton is essentially a time table for a digital machine (or a digital controller), describing what the machine should do at a given time point, and it can be easily transformed to an executable program. Finally, we present a method to check whether a time-triggered automaton recognizes the language of a timed automaton according to the new semantics.

1 Introduction

Timed automata [AD94] have been recognized as a basic model for real time systems. A number of tools based on timed automata have been developed (e.g. [BDM+98,LPY97]), and applied successfully to model and verify industrial systems (e.g. [BGK+96,DY00]). A recent line of work uses timed automata for the schedulability analysis of tasks in real time systems [FPY02,KY04,WH04]. The main idea here is to use timed automata to describe the arrival patterns of external events triggering real-time tasks. One implicit assumption in this line of work and its extensions to synthesize executable code [AFP+02] is that the arrival sequences of events described by the automaton can be admitted *instantly* by the interrupt-processing system of the given platform. However, this is not realistic in realizations involving a digital computer driven by a system clock with a fixed granularity of time.

Therefore, we wish to study the notion of timely executions of task arrival patterns in settings where the implementation model is explicitly clock-driven. Time-triggered architecture [KB01,Kop98] is one such well-recognized implementation paradigm for the design of safety-critical real-time systems, a programming language Giotto [HKH03] provides an abstract model for implementation

P. Gardner and N. Yoshida (Eds.): CONCUR 2004, LNCS 3170, pp. 340–354, 2004.

of time-triggered systems. In a time-triggered system, computation steps are controlled by one global clock; at each time moment only actions enabled by the clock value can be taken. Thus, computations are time-deterministic in such systems.

We present a (finite state) model of computations running on time-triggered architectures called *time-triggered automata* accepting digitalized versions of timed languages. A time-triggered automaton may take transitions (to consume external events) only at integer time points determined by periodic and non-periodic timing constraints denoted $a(n)$ and $a[n]$ respectively. A transition labeled with $a(n)$ means that every n time units, it should check if a has occurred; a transition labeled with $a[n]$ means that in n time units from now, it should check if a has occurred. Our time-triggered automata capture only the basic aspect of time-triggered architectures. In the present setting they could as well be called "digital automata". However, we consider time-triggered architectures and protocols to be important and the model we propose here is a simple but potentially useful first step towards formalizing this framework in an automata theoretic setting. A time-triggered automaton is essentially a timetable for a digital controller, describing what the controller should do at a given time point, and it can be easily transformed to an executable program. Time-triggered automata can be viewed as special kinds of timed automata. However, their intended role is to capture the behavior of digital machines and their syntax is geared towards achieving this in a transparent way.

Our main goal is to semantically tie together the timed-automata based specifications to the time-triggered-automaton based implementations. We do so by associating a deadline parameter ϵ with the timed automaton. The idea is that there is a buffer[1] into which the event arrivals generated by the timed automaton will be put and each such event will be observable for ϵ time units from the time it was released (i.e. put into the buffer). A correct implementation mechanism is then required to pick up each event before its deadline expires. The parameter ϵ associated with the timed automaton and the unit of time associated with the time-triggered automaton are with reference to the same global time scale. Further, both types of automata share a common alphabet of events. These two components provide the handle for semantically relating the language of the timed automaton to that of the time-triggered automaton.

More specifically, given a timed automaton TA and the deadline parameter ϵ we extract the ϵ-digitalized timed language of TA. In loose terms, this language is obtained by taking a timed trace of TA and converting the timestamp t of each event into an integer-valued time k_t provided $k_t - t$ does not exceed ϵ. This captures the idea that an event released at time t can be, in principle, picked up at integer time k_t before the deadline associated with this release has expired.

We then associate a digitalized timed language with the corresponding time-triggered automaton TTA and formulate the notion of TTA correctly accepting

[1] For ease of presentation we consider the buffer to be FIFO, but the same ideas apply also to random access buffer modeled as a multiset.

ϵ-digitalization of the timed language of TA. This notion is intended to capture the processing by TTA of the request patterns generated by TA. Each event generated by TA will be put in the buffer with an associated timer set to ϵ. At integer time points dictated by its transition relation, the TTA will pick up a specified sequence of events from the buffer provided none of these events have an expired timer. This whole process fails anytime an event misses its deadline before it gets picked by the TTA. In order to simplify the presentation we have identified the servicing of the request represented by an event with being picked up by the TTA in time. In other words, we have abstracted away the tasks associated with the events and their timely executions on the chosen platform which may be limited resource-wise. Further, we assume the same fixed observability period (i.e. deadline) for each event.

Finally, we show that it is possible to effectively decide whether a TTA correctly services the request patterns generated by the TA. The proof technique is an adaptation of the timed automaton based technique used in [FPY02] for solving schedulability problems.

1.1 Related Work

In the literature on timed automata, there has been a considerable amount of work related to discretization issues, e.g. [HMP92,LY97,GHJ97] where just the timed traces with integral timestamps of events are used to characterize the real time semantics, and to identify (digitizable) dense-time properties that can be checked based on the integral traces. A recent study on digitalization is [OW03] which is also a very good source for other related work. Despite the overlap in terminology the key difference is that here we do not consider the *actual* timed traces of a timed automaton. Rather, we study in what sense the timed traces of an automata can be recognized by a digital computer (i.e. a digital controller). The controller should consume (or record) all the timed traces of a timed automaton, not only those in which the timestamps of events are integers. Due to the non-instant observability of events, this consumption can take place at any integer time point shortly (within ϵ time units) after the event has occurred.

The idea of checking whether all events generated by an environment are picked-up by a synchronous program sufficiently fast has been also studied in [BPS00,BCP$^+$01,CPP$^+$01,STY03]. Here authors use Esterel programs annotated with temporal constraints as event handlers to consume events. A timed automata model of the application can be automatically generated out of such a program. Then the behavior of the model is checked against the environment (modeled again in Esterel with added non-deterministic primitive and then translated to a timed automaton). The presented methodology has been implemented in the tool Taxys [CPP$^+$01] and successfully used for industrial applications. Our focus here is on the basic semantic issues that arise in the interaction between timed automata and digital machines and our study is carried out in a language-independent automata theoretic setting. We consider timed automata as timed specifications and study in what sense such a specification involving dense time is implemented by a digital machine that is described as

time-triggered automaton. For $\epsilon = 1$ our correctness checking coincides with checking of throughput constraint satisfaction described in [CPP+01].

Recently, a new semantics for timed automata geared towards their implementation was proposed in [WDR04]. This work is also based on non-instant observability of events. Its main goal is to detect timed automata where two consecutive actions are taken shortly one after another. When this is not the case, the timed automaton can be implemented as an event-driven controller using sampling. In contrast with our semantics, this work deals with real timed traces. Also the implementation simulates timed automaton specification of a controller whereas in our case there is no timed automata model of a controller (in our case, the environment in the only specification).

The rest of the paper is organized as follows: Section 2 contains a brief introduction to timed automata and a detailed description of the ϵ-semantics and gives two alternative formalizations of digitalization. The syntax and semantics of time-triggered automata is defined in Section 3. Section 4 presents a correctness criterion for time-triggered automata and our technique for verifying correctness. Section 5 concludes the paper.

2 Timed Automata: ϵ-Semantics and Digitalization

In the following, we briefly introduce the standard semantics for timed automata. To capture the non-instant observability of timed events, we present a new semantics for timed automata. It gives rise to a simple and natural notion of digitalization for timed languages.

2.1 Timed Automata

A timed automaton [AD94] is a standard finite-state automaton extended with a finite collection of real-valued clocks. Events are accepted by an automaton at real-time points satisfying given timing constraints. Assume a set of real-valued clocks denoted by $\mathcal{C}$ and a finite alphabet Σ whose elements represent events.

Definition 1. *A* timed automaton $\mathcal{A}_{TA}$ *is a tuple* $\langle \mathcal{C}, \Sigma, N, l_0, E \rangle$ *where*

- $\mathcal{C}$ *is a set of real-valued clocks,*
- Σ *is a finite alphabet of events,*
- N *is a finite set of locations,*
- $l_0 \in N$ *is the initial location, and*
- $E \subseteq N \times \Phi(\mathcal{C}) \times \Sigma \times 2^{\mathcal{C}} \times N$ *is the set of edges (describing possible transitions).*

The set $\Phi(\mathcal{C})$ of clock constraints ϕ is defined as a set of conjunctive formulas of atomic constraints in the form: $x_i \sim m$ or $x_i - x_j \sim n$ where $x_i, x_j \in \mathcal{C}$ are clocks, $\sim \in \{\leq, <, \geq, >\}$, and m, n are natural numbers. A clock valuation $u \in [\mathcal{C} \to \mathbb{R}_{\geq 0}]$ is a function mapping clocks to non-negative real numbers. We use $u + t$ to denote the clock assignment which maps each clock x to the value

$u(x) + t$, and $u[r \mapsto 0]$ for $r \subseteq \mathcal{C}$ to denote the clock assignment which maps each clock in r to 0 and agrees with u for the other clocks (i.e. $\mathcal{C} \backslash r$). An edge (l_1, ϕ, e, r, l_2) represents a transition from location $l_1 \in N$ to location $l_2 \in N$ accepting an input symbol (we will call it an *event*) $e \in \Sigma$, and resetting clocks in $r \subseteq \mathcal{C}$ to zero, if the current values of clocks satisfy ϕ.

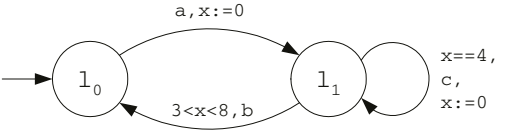

Fig. 1. An example timed automaton

An example timed automaton is shown in Figure 1. It consists of the set of locations $N = \{l_0, l_1\}$ where l_0 is the initial location, the set of clocks $\mathcal{C} = \{x\}$, the alphabet of events $\Sigma = \{a, b, c\}$, and the set of clock constraints $\Phi(\mathcal{C}) = \{x \geq 4, x \leq 4, x > 3, x < 8\}$.

A *timed event* is a pair (t, e), where $e \in \Sigma$ is an event accepted by $\mathcal{A}_{TA}$ after $t \in \mathbb{R}_{\geq 0}$ time units since $\mathcal{A}_{TA}$ has been started. This absolute time t is called a *timestamp* of the event e. A *timed trace* is a (possibly infinite) sequence of timed events $\xi = (t_1, e_1)(t_2, e_2)...$ over events $e_1, e_2, ...$ associated with the corresponding timestamps $t_1, t_2, ...$ where $\forall i \geq 1\colon\ t_i \leq t_{i+1}$.

Definition 2. *A* run *of a timed automaton $\mathcal{A}_{TA} = \langle \mathcal{C}, \Sigma, N, l_0, E \rangle$ over a timed trace $\xi = (t_1, e_1)(t_2, e_2)(t_3, e_3)...$ is a (possibly infinite) sequence of the form*

$$(l_0, u_0) \xrightarrow[t_1]{e_1} (l_1, u_1) \xrightarrow[t_2]{e_2} (l_2, u_2) \xrightarrow[t_3]{e_3} \dots$$

where $l_i \in N$, u_i is a clock valuation, satisfying the following conditions:

- *$u_0(x) = 0$ for all $x \in \mathcal{C}$.*
- *for all $i \geq 1$, there is an edge $(l_{i-1}, \phi_i, e_i, r_i, l_i)$ such that $(u_{i-1} + t_i - t_{i-1})$ satisfies ϕ_i and $u_i = (u_{i-1} + t_i - t_{i-1})[r_i \mapsto 0]$.*

The timed language *$L(\mathcal{A}_{TA})$ over alphabet Σ is the set of all timed traces ξ for which there exists a run of $\mathcal{A}_{TA}$ over ξ.*

For example, the timed automaton shown in Figure 1 can run over the following timed trace: $\xi = (0.5, a)(4.5, c)(7.6, b)(7.9, a)\dots$

2.2 The ϵ-Semantics and Digitalization

One can interpret a timed automaton as the model of an environment producing events. For example, at some real-time points a plant can produce events to which a controller should respond. It is a natural assumption that each such event remains observable for a short time interval during which it can be picked up

by a controller. However, the non-instant observability of events is not reflected in the standard semantics for timed-automata. We introduce the *ϵ-semantics* to capture the fact that each event remains observable for ϵ time units after its occurrence.

Assume that an event occurring at time t is put in a communication buffer between the environment and the controller and remains there until consumed by the controller or expired at $t + \epsilon$. We introduce a queue q to represent the buffer between the environment and the controller. Elements of q are pairs (e, δ) where $e \in \Sigma$ is an event and $\delta \in \mathbb{R}$ is the relative deadline of e in q. We denote $q :: e$ the queue q with a pair (e, ϵ) inserted in the back of it, and $q \backslash e$ the queue q with (e, δ) removed from it where (e, δ) is the head of the queue q. We write $q - d$ for the queue q in which deadlines in all pairs are decreased by d. The states of a timed automaton in the ϵ-semantics are in the form: (l, u, q, τ) where

- l is the current location,
- u is the current clock valuation,
- q is the current event queue, and
- τ is the current time i.e. the global time elapsed since the automaton has been started in the initial state.

For simplicity, we assume that all events are associated with a fixed constant $\epsilon \in \mathbb{N}_{>0}$ (by $\mathbb{N}_{>0}$ we denote the set of positive natural numbers). However, the setting can be easily extended to allow the constant to be a positive rational number.

Definition 3. *Let $\epsilon \in \mathbb{N}_{>0}$ be a positive natural number. The ϵ-semantics of a timed automaton $\langle \mathcal{C}, \Sigma, N, l_0, E \rangle$ with initial state (l_0, u_0, q_0, τ_0), where q_0 is an empty queue and $\tau_0 = 0$, is a labeled transition system $\mathcal{S}^\epsilon$ defined by the following rules:*

- Consumption: $(l, u, q, \tau) \xrightarrow{e} (l, u, q \backslash e, \tau)$ *if* $(e, \delta) = \mathsf{Head}(q)$ *such that* $\epsilon \geq \delta > 0$,
- Production: $(l_1, u_1, q, \tau) \longrightarrow (l_2, u_2, q :: e, \tau)$ *if* $\exists (l_1, \phi, e, r, l_2) \in E$ *such that* u_1 *satisfies* ϕ *and* $u_2 = u_1[r \mapsto 0]$.
- Delay: $(l, u, q, \tau) \xrightarrow{d} (l, u + d, q - d, \tau + d)$.

If an event has not been consumed after ϵ time units by means of consumption transition then no other consequent event would be consumed later. Such a state is considered as a failure of a controller to consume all events. Note also that environments are allowed to produce infinitely many events in a finite time. Naturally, a controller for such environments cannot be implemented.

In order to describe how events produced by a timed automaton are consumed by a digital machine, we introduce a notion of a digitalized timed trace. In the ϵ-semantics several events can be consumed at the same time but only in the order they have been produced, i.e. the controller cares about dependencies between events.

Definition 4. *An* ϵ-digitalized timed trace ξ^ϵ *in* $\mathcal{S}^\epsilon$ *is a (possibly infinite) sequence* $(t_1^\epsilon, e_1), (t_2^\epsilon, e_2), (t_3^\epsilon, e_3), \ldots$ *such that there exists a path in* $\mathcal{S}^\epsilon$ *where*

$e_1, e_2, e_3, \ldots$ are labels on the consumption transitions in the order in which they occur on the path, $t^\epsilon_1, t^\epsilon_2, t^\epsilon_3, \ldots$ are their absolute timestamps respectively, and $t^\epsilon_i \in \mathbb{N}_{>0}$ for all i.

The ϵ-digitalized timed language *$L^\epsilon(\mathcal{A}_{TA})$ over alphabet Σ according to the ϵ-semantics is the set of all ϵ-digitalized timed traces ξ^ϵ in $\mathcal{S}^\epsilon$ for $\mathcal{A}_{TA}$.*

We shall see (in the following subsection) that for $\epsilon > 1$ each run of $\mathcal{A}_{TA}$ can have several corresponding ϵ-digitalized timed traces in which the distance between the real-valued timestamp and corresponding digitalized timestamp for each event is limited by ϵ. For the case when $\epsilon = 1$, there is only one such ϵ-digitalized timed trace for each run of $\mathcal{A}_{TA}$. This is a useful property of our notion of digitalization. It means that any sequence of events with timestamps in the standard semantics will be caught by at least one digitalized trace. This enables to formulate the correctness criterion as a language inclusion property.

2.3 An Alternative Characterization

We present an alternative characterization of the ϵ-digitalized timed language for timed automata. This characterization establishes a connection between a timed trace and its ϵ-digitalized versions. In the following we use rounded-up time points $\lceil t \rceil$ where $\lceil t \rceil$ denotes the least integer such that $t \leq \lceil t \rceil$.

Definition 5. *For a timed trace $\xi = (t_1, e_1)(t_2, e_2)(t_3, e_3) \ldots$, an ϵ-rounded-up timed trace $\lceil \xi \rceil^\epsilon$ is a (possibly infinite) sequence $(t'_1, e_1)(t'_2, e_2)(t'_3, e_3) \ldots$, such that there exists a sequence $k_1, k_2, k_3, \ldots$, where $k_i \in \{0, \ldots, \epsilon-1\}$, for all $i \geq 1$, $t'_i = \lceil t_i \rceil + k_i$, and $t'_1, t'_2, t'_3, \ldots$ is a non-decreasing sequence of timestamps.*

The ϵ-rounded-up timed language $\lceil L(\mathcal{A}_{\mathcal{T}\mathcal{A}}) \rceil^\epsilon$ over alphabet Σ is the set of all ϵ-rounded-up timed traces $\lceil \xi \rceil^\epsilon$ where $\xi \in L(\mathcal{A}_{TA})$.

For $\epsilon = 1$, all k_i are equal to 0, and 1-rounded-up timed trace can be constructed just by rounding-up all timestamps of all timed events. Moreover, there is just one 1-rounded-up timed trace $\lceil \xi \rceil^1$ for each timed trace ξ. For example, for the timed trace $\xi = (0.5, a)(4.5, c)(7.6, b)(7.9, a) \ldots$, the 1-rounded-up timed trace is $\lceil \xi \rceil^1 = (1, a)(5, c)(8, b)(8, a) \ldots$

Intuitively, an event e_i occurring at a real-valued time point t_i should remain observable by the controller until the closest integral time point $\lceil t_i \rceil$. Therefore, all events with timestamp t_i, such that $\lceil t_i \rceil - \epsilon < t_i \leq \lceil t_i \rceil$, are consumed by a digital machine at the nearest greater (or equal) integer point $\lceil t_i \rceil$.

For $\epsilon > 1$, there are several ϵ-rounded-up timed traces for each timed trace. Each ϵ-rounded-up timed trace describes at which (integer) time point each event is consumed. A timed event (t_i, e_i) can be consumed at any time point $t'_j = \lceil t_i \rceil + k$, where $k \in \{0, \ldots, \epsilon-1\}$. Also, for each two timed events (t_1, e_1) and (t_2, e_2) in the same timed trace where $t_1 < t_2$ the event e_1 must be handled before the event e_2. For example, for $\epsilon = 2$ and the timed trace ξ, we have two examples of digitalized timed traces: $\lceil \xi \rceil^2_1 = (2, a)(6, c)(9, b)(9, a) \ldots$ and $\lceil \xi \rceil^2_2 = (1, a)(6, c)(8, b)(9, a) \ldots$

In the following proposition we state the equivalence of ϵ-digitalized language and ϵ-rounded-up language. It will allow us to use ϵ-rounded-up timed traces instead of ϵ-digitalized ones and to exploit the fact that the former can be obtained from a timed trace in a natural way by rounding-up the timestamps of all its events.

Theorem 1. *For each timed automaton* $\mathcal{A}_{TA}$, $L^{\epsilon}(\mathcal{A}_{TA}) = \lceil L(\mathcal{A}_{TA}) \rceil^{\epsilon}$.

Proof. By induction on the length of the timed trace accepted by $\mathcal{A}_{TA}$. □

3 Time-Triggered Automata

In this section, we present an automaton model for time-triggered systems and define the digitalized languages they accept. Time-triggered systems, in contrast to event-driven systems, run according to a given timetable. They have a global reference clock and the behavior of the system at each time point is determined by a part of the timetable corresponding to the value of the global clock. We want to remark again that time-triggered automata can be easily represented as timed automata and we introduce them only to capture the behavior of digital machines syntactically in a transparent way.

Similar to finite automata, time-triggered automata are finite labeled directed graphs. An input alphabet for these automata is a finite set of finite event sequences. We will refer to each such sequence as a *request*. We could use single events instead of sequences but this would make it awkward for the automata to be able to read several events at one integer time point. Each transition of a time-triggered automaton is labeled with a request and a time constraint from $\Theta = \{[n], (n) | n \in \mathbb{N}_{>0}\}$, where $[n]$ (read as "at n") denotes instantaneous (non-periodic) constraints and (n) (read as "every n time units") denotes periodic constraints. Intuitively, constraints specify a pattern of time points at which requests are accepted by a time-triggered automaton. If a request is specified together with a periodic constraint then the automaton will check every n time units whether it is on the input. Instantaneous constraints determine only a single time point for handling the corresponding request.

Definition 6. *A* time-triggered automaton $\mathcal{A}_{TTA}$ *is a tuple* $\langle \Sigma, S, s_0, T \rangle$ *where*

- Σ *is a finite alphabet of events,*
- S *is a finite set of locations,*
- $s_0 \in S$ *is the initial location, and*
- $T \subseteq S \times \Sigma^* \times \Theta \times S$ *is a finite set of edges.*

We will use $s_1 \xrightarrow{\omega[n]} s_2$ *to denote* $(s_1, \omega, [n], s_2) \in T$ *and* $s_1 \xrightarrow{\omega(n)} s_2$ *to denote* $(s_1, \omega, (n), s_2) \in T$. *We use symbol null to denote an empty sequence of events (sequence of length* 0*).*

Figure 2 shows a time-triggered automaton recognizing the digitalized language of the timed automaton in Figure 1. For example, in state S_0, every 1 time

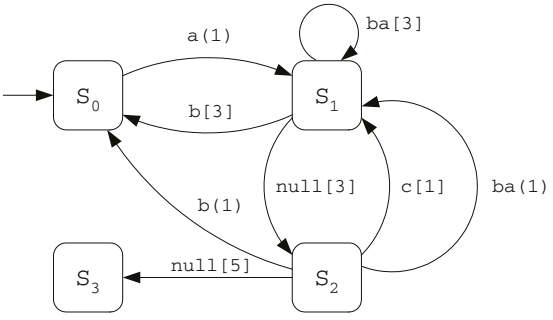

Fig. 2. An example time-triggered automaton

unit, the automaton checks if a has occurred, and if a is observed, it consumes a and moves to S_1. In S_1, if ba is observed at time 3 (since it entered S_1), it consumes ba and moves back to S_1. If nothing is observed (represented by *null*) in S_1 at time 3, the automaton moves to S_2.

Time-triggered automata are non-deterministic in general. However, in order to make them implementable one has to deal with the deterministic subclass which can be obtained by prohibiting the following pairs of transitions outgoing from the same location:

- $s_1 \xrightarrow{\omega[n]} s_2$, $s_1 \xrightarrow{\omega[m]} s_3$ where $n = m$,
- $s_1 \xrightarrow{\omega[n]} s_2$, $s_1 \xrightarrow{\omega(m)} s_3$ where $m \bmod n = 0$, and
- $s_1 \xrightarrow{\omega(n)} s_2$, $s_1 \xrightarrow{\omega(m)} s_3$.

From now on we will consider only deterministic time-triggered automata.

3.1 Semantics

Generally, time-triggered automata can make the same types of transitions as timed automata, i.e. they can either change the location performing an action or delay in the same location. An automaton may change the location only when a transition outgoing from this location is enabled. Transitions are enabled at certain time points determined by their time constraints and the global time. Each edge with instantaneous constraint n is *enabled* when exactly n time units elapsed since the automaton has entered the location. Each edge with periodic constraint n is *enabled* every n time units after the automaton has entered the location.

A state of the time-triggered automaton $\mathcal{A}_{TTA}$ is a pair (s, t) where $s \in S$ is a location in $\mathcal{A}_{TTA}$ and $t \in \mathbb{R}_{\geq 0}$ is the time since $\mathcal{A}_{TTA}$ has entered s.

Definition 7. *The semantics of a time-triggered automaton $\langle S, \Sigma, s_0, T\rangle$ with initial state $(s_0, 0)$ is a labeled transition system defined by the following rules:*

- $(s_1, t) \xrightarrow{\omega} (s_2, 0)$ *if* $s_1 \xrightarrow{\omega[n]} s_2$ *and* $n = t$
- $(s_1, t) \xrightarrow{\omega} (s_2, 0)$ *if* $s_1 \xrightarrow{\omega(m)} s_2$ *and* $t \bmod m = 0$, $t > 0$
- $(s, t) \longrightarrow (s, t + d)$ *if* $t + d \leq \lfloor t + 1 \rfloor$

The condition $t + d \leq \lfloor t + 1 \rfloor$, where $\lfloor n \rfloor$ is the greatest integer such that $\lfloor n \rfloor \leq n$, in the third rule ensures that no integer time point can be skipped by a delay transition. Allowing real-valued delays is not necessary here, but it will help us later when we will compose timed automata and time-triggered automata together. Note, that in the transition system for a deterministic $\mathcal{A}_{TTA}$ there cannot be a state (s, t) with two outgoing transitions labeled by the same request ω. Now we can define a run of $\mathcal{A}_{TTA}$ over a digital timed trace (time trace with integral timestamps only). Let $H(t, \omega) = (t, e_1)(t, e_2) \dots (t, e_n)$ where $\omega = e_1 e_2 \dots e_n$ and $t \in \mathbb{N}_{>0}$ be the function unrolling a request associated with the timestamp t into a digital timed trace.

Definition 8. *A run of a time-triggered automaton* $\mathcal{A}_{TTA} = \langle S, \Sigma, s_0, T \rangle$ *over a digital timed trace* $\lceil \xi \rceil = (t_1, e_1)(t_2, e_2)(t_3, e_3) \dots$ *is a sequence of the form*

$$s_0 \xrightarrow[t'_1]{\omega_1} s_1 \xrightarrow[t'_2]{\omega_2} s_2 \xrightarrow[t'_3]{\omega_3} \dots$$

where s_i *is a location in* $\mathcal{A}_{TTA}$*, satisfying the following requirements:*

- $H(t'_1, \omega_1) H(t'_2, \omega_2) \dots = (t_1, e_1)(t_2, e_2)(t_3, e_3) \dots = \lceil \xi \rceil$
- *for all* $i \geq 1$ *there is a transition* $(s_{i-1}, t'_i - t'_{i-1}) \xrightarrow{\omega_i} (s_i, 0)$ *and there is no transition* $(s_{i-1}, t') \xrightarrow{null} (s', 0)$, $t' < t'_i - t'_{i-1}$ *in the transition system induced by* $\mathcal{A}_{TTA}$*.*

The language $L(\mathcal{A}_{TTA})$ *is the set of all digital timed traces* $\lceil \xi \rceil$ *for which there exists a run of* $\mathcal{A}_{TTA}$ *over* $\lceil \xi \rceil$*.*

The first requirement says that $\mathcal{A}_{TTA}$ must consume all timed events at the correct time and in the specified order. We consider consuming a request (a sequence of events) as consuming several events at the same time point, but in the order in which they appear in the request. The second requirement specifies when a request can be consumed. Note, that if only a transition labeled by *null* is enabled then it must be taken immediately. By this, we want to ensure deterministic behavior of $\mathcal{A}_{TTA}$.

4 Correctness Checking

We now show in which sense a time-triggered automaton $\mathcal{A}_{TTA}$ handles correctly the events produced by a timed automaton $\mathcal{A}_{TA}$ where each event expires within ϵ time units after having been released.

For time-triggered automata we apply maximal progress assumption, i.e. if a particular transition of $\mathcal{A}_{TTA}$ is enabled and there is a corresponding request

(sequence of events produced by $\mathcal{A}_{TA}$) on the input it must be performed immediately. This gives us temporal determinism of the composition of $\mathcal{A}_{TA}$ and $\mathcal{A}_{TTA}$. $\mathcal{A}_{TTA}$ does not have to guess whether to perform an action or whether to leave the events in the buffer and perform this action at the next time tick.

Ideally we want that for each timed trace of $\mathcal{A}_{TA}$, the corresponding ϵ-rounded-up timed trace should be accepted by $\mathcal{A}_{TTA}$. The problem is that when $\epsilon > 1$, there are several ϵ-rounded-up timed traces $\lceil \xi \rceil^{\epsilon}$ for each timed trace ξ. For example, the $\mathcal{A}_{TA}$ shown in Figure 3(a) has a run over the timed trace $\xi = (0.5, a)(5, b)$. Given $\epsilon = 2$, there are several ϵ-rounded-up traces for ξ, for instance $\lceil \xi \rceil^2_1 = (1, a)(5, b)$ and $\lceil \xi \rceil^2_2 = (2, a)(5, b)$. Let us assume that we are given the $\mathcal{A}_{TTA}$ shown in Figure 3(b) which can run over $\lceil \xi \rceil^2_2$ but cannot run over $\lceil \xi \rceil^2_1$. However, according to the maximal progress assumption, as a appears in ξ at 0.5, it should be picked up by $\mathcal{A}_{TTA}$ no later than at 1. Unfortunately, $\mathcal{A}_{TTA}$ does not accept $\lceil \xi \rceil^2_1$ which means that it is not correct with respect to $\mathcal{A}_{TA}$ and ϵ (even if there exists an ϵ-rounded-up timed trace for ξ which is accepted by $\mathcal{A}_{TTA}$). Thus, for given ξ and ϵ we want to select just those ϵ-rounded-up timed traces which correspond to the maximal progress assumption for $\mathcal{A}_{TTA}$, namely $\lceil \xi \rceil^2_1$, and check whether they are accepted by $\mathcal{A}_{TTA}$.

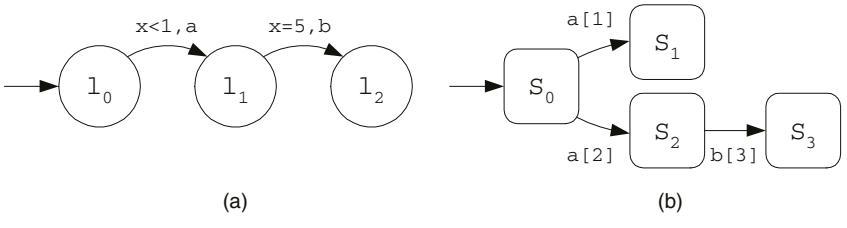

Fig. 3. A timed automaton (a) and a time-triggered automaton (b)

Definition 9. *Given $\mathcal{A}_{TTA}$ and a finite timed trace ξ, we define the notion of $\mathcal{A}_{TTA}$* promptly *accepting an ϵ-rounded-up timed trace $\lceil \xi \rceil^{\epsilon}$ inductively.*

1. *If $|\xi| = 0$ ($|\xi|$ is the length of ξ) then $\lceil \xi \rceil^{\epsilon}$ is accepted promptly.*
2. *If $\xi = \xi' \cdot (t, e)$ then $\lceil \xi \rceil^{\epsilon} = \lceil \xi' \rceil^{\epsilon} \cdot (t', e)$ is accepted promptly iff $\lceil \xi' \rceil^{\epsilon}$ is accepted promptly, $\lceil \xi \rceil^{\epsilon}$ is accepted by $\mathcal{A}_{TTA}$, and no ϵ-rounded-up timed trace $\lceil \xi' \rceil^{\epsilon} \cdot (t'', e)$ for ξ where $t'' < t'$ is also accepted by $\mathcal{A}_{TTA}$.*

Finally we say that (possibly infinite) sequence $\lceil \xi \rceil^{\epsilon}$ is accepted promptly iff all its finite prefixes are accepted promptly.

Now we can define our notion of correctness.

Definition 10. *Given $\epsilon \in \mathbb{N}_{>0}$, a timed automaton $\mathcal{A}_{TA}$, and a time-triggered automaton $\mathcal{A}_{TTA}$, we say that $\mathcal{A}_{TTA}$ is* correct *with respect to $\mathcal{A}_{TA}$ and ϵ iff for each timed trace $\xi \in L(\mathcal{A}_{TA})$ there exists an ϵ-rounded-up timed trace $\lceil \xi \rceil^{\epsilon}$ which is promptly accepted by $\mathcal{A}_{TTA}$.*

As ϵ-rounded-up timed trace specifies all servicing time points within the deadline and prompt acceptance corresponds to a possible run of $\mathcal{A}_{TTA}$ (satisfying the maximal progress assumption) in which all events are picked-up in specified time points, the above definition ensures that no deadline will be missed.

For the case $\epsilon = 1$, it follows at once that the correctness criterion can be captured by a simple language inclusion property as stated below.

Proposition 1. *For $\epsilon = 1$, a time-triggered automaton $\mathcal{A}_{TTA}$ is* correct *with respect to a timed automaton $\mathcal{A}_{TA}$ and ϵ iff $\lceil L(\mathcal{A}_{TA})\rceil^{\epsilon} \subseteq L(\mathcal{A}_{TTA})$.*

Proof. For $\epsilon = 1$, prompt acceptance coincides with (simple) acceptance and for each timed trace ξ there exists exactly one ϵ-rounded-up timed trace $\lceil\xi\rceil^{\epsilon}$. According to Definition 5, $\lceil L(\mathcal{A}_{TA})\rceil^{\epsilon} = \{\lceil\xi\rceil^{\epsilon} | \ \xi \in \text{Ł}(\mathcal{A}_{TA})\}$. □

We now propose an effective method (which can in fact be automated) to verify the correctness of $\mathcal{A}_{TTA}$ with respect to $\mathcal{A}_{TA}$ and ϵ. Our strategy is to reduce this verification problem to the problem of checking schedulability for timed automata extended with asynchronous tasks.

In [FPY02] a classical notion of schedulability has been extended to timed automata asynchronously releasing tasks for execution when discrete transitions are taken. Tasks have to complete their execution before the deadline. A processing unit (e.g. CPU) executes the released tasks stored in a queue according to some scheduling policy (e.g. FPS or EDF). The goal of the schedulability analysis is then to check that all the released tasks meet their deadlines along all possible runs of an automaton.

Since we consider $\mathcal{A}_{TTA}$ as a device which handles requests that come from $\mathcal{A}_{TA}$, it is natural to interpret $\mathcal{A}_{TA}$ as a timed automaton extended with asynchronous tasks. Each request produced by $\mathcal{A}_{TA}$ corresponds to a task annotated with the relative deadline equal to ϵ. When released by $\mathcal{A}_{TA}$, a task is put in the task queue, which we will denote Ω. In its turn, $\mathcal{A}_{TTA}$ can be seen as a processing unit (Run-function in [FPY02]) that executes the ready queue of tasks according to a scheduling strategy (First Come First Served (FIFO), in our case). The setting used to perform an automatic verification of the correctness of $\mathcal{A}_{TTA}$ with respect to $\mathcal{A}_{TA}$ and ϵ is shown in Figure 4.

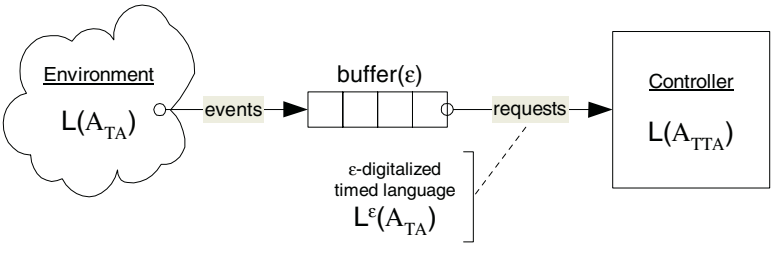

Fig. 4. The setting for verification of correctness of $\mathcal{A}_{TTA}$ w.r.t. $\mathcal{A}_{TA}$,ϵ

We define a machine $\mathbb{M}_{\mathcal{A}_{TA},\epsilon,\mathcal{A}_{TTA}}$ used for automatic correctness checking of $\mathcal{A}_{TTA}$ with respect to $\mathcal{A}_{TA}$ and ϵ as a triple $\langle \mathcal{A}_{TA}, \epsilon, \mathcal{A}_{TTA} \rangle$. A state of $\mathbb{M}_{\mathcal{A}_{TA},\epsilon,\mathcal{A}_{TTA}}$ is (l, u, q, s, t) where l is a location in $\mathcal{A}_{TA}$, u is a clock assignment, q is a state of the task queue Ω, s is a location in $\mathcal{A}_{TTA}$, and $t \in \mathbb{R}_{\geq 0}$ is the time elapsed since $\mathcal{A}_{TTA}$ has entered s.

We denote $q :: e$ the queue with a pair (e, ϵ) inserted in the back of the queue, and $q \backslash \omega$, $\omega = e_1 e_2 \dots e_n$ the queue with $(e_1, \delta_1), (e_2, \delta_2), \dots, (e_n, \delta_n)$ removed from it, where $(e_1, \delta_1), (e_2, \delta_2), \dots, (e_n, \delta_n)$ are the n foremost pairs in the queue ($\mathsf{Head}(q) = (e_1, \delta_1)$). Otherwise, it is not possible to perform $q \backslash \omega$. We remove events from the queue in the order they have been inserted (FIFO).

Definition 11. *The semantics of* $\mathbb{M}_{\mathcal{A}_{TA},\epsilon,\mathcal{A}_{TTA}}$*, where* $\mathcal{A}_{TA} = \langle \mathcal{C}, \Sigma, N, l_0, E \rangle$, $\mathcal{A}_{TTA} = \langle \Sigma, S, s_0, T \rangle$*, with initial state* $(l_0, u_0, q_0, s_0, 0)$ *is a labeled transition system defined by the following rules:*

- $(l, u, q, s, t) \xrightarrow{e} (l', u[r \mapsto 0], q :: e, s, t)$*, if* $(l, \phi, e, r, l') \in E$*, and* $u \models \phi$*,*
- $(l, u, q, s, t) \longrightarrow (l, u, q \backslash \omega, s_1, 0)$*, if* $t \in \mathbb{N}_{>0}$*, it is possible to perform* $q \backslash \omega$*, and* $s \xrightarrow{\omega[t]} s_1$*, or* $s \xrightarrow{\omega(n)} s_1$ *where* $t \bmod n = 0$*,*
- $(l, u, q, s, t) \xrightarrow{d} (l, u + d, q - d, s, t + d)$*, if* $t + d \leq \lfloor t + 1 \rfloor$*, and if* $t \in \mathbb{N}_{>0}$ *then there is no enabled edge labeled with request* ω *in* $\mathcal{A}_{TTA}$ *such that it is possible to perform* $q \backslash \omega$*.*

A state (l, u, q, s, t) of $\mathbb{M}_{\mathcal{A}_{TA},\epsilon,\mathcal{A}_{TTA}}$ is called *faulty* if for some pair $(e, \delta) \in q$, an event e misses its deadline, i.e. $\delta \leq 0$. $\mathbb{M}_{\mathcal{A}_{TA},\epsilon,\mathcal{A}_{TTA}}$ is called *schedulable* iff no faulty state is reachable from the initial state of $\mathbb{M}_{\mathcal{A}_{TA},\epsilon,\mathcal{A}_{TTA}}$.

Theorem 2. *$\mathcal{A}_{TTA}$ is correct with respect to $\mathcal{A}_{TA}$ and ϵ iff $\mathbb{M}_{\mathcal{A}_{TA},\epsilon,\mathcal{A}_{TTA}}$ is schedulable. Further, one can effectively decide the schedulability of $\mathbb{M}_{\mathcal{A}_{TA},\epsilon,\mathcal{A}_{TTA}}$.*

Sketch of Proof. The first part of the result follows from the construction of $\mathbb{M}_{\mathcal{A}_{TA},\epsilon,\mathcal{A}_{TTA}}$. According to Definition 10 we are looking for ϵ-rounded-up timed traces promptly accepted by $\mathcal{A}_{TTA}$ for each run of $\mathcal{A}_{TA}$. As follows from Proposition 1 and Definition 3, each such ϵ-rounded-up timed trace corresponds to a path in the labeled transition system $\mathcal{S}^{\epsilon}$ for $\mathcal{A}_{TA}$. We can view prompt acceptance of such a word as a path in the transition system obtained as the synchronous product of $\mathcal{S}^{\epsilon}$ and the transition system induced by $\mathcal{A}_{TTA}$. Now, production, consumption, and delay transitions in this product correspond to the transitions of $\mathbb{M}_{\mathcal{A}_{TA},\epsilon,\mathcal{A}_{TTA}}$, respectively. Faulty states in $\mathbb{M}_{\mathcal{A}_{TA},\epsilon,\mathcal{A}_{TTA}}$ correspond to states in $\mathcal{S}^{\epsilon}$ where event consumption failed.

The second part of the result boils down to a straightforward adaptation of the decidability argument for the schedulability of timed automata extended with tasks developed in [FPY02]. In this work, a timed automaton extended with tasks, the task queue, the Run-function, and a scheduling policy are encoded as a timed automaton and the schedulability problem is reduced to the reachability of a predefined faulty state. As $\mathcal{A}_{TTA}$ corresponding to the Run-function is deterministic, it can be easily encoded in this timed automaton instead of the Run-function. □

5 Conclusions

In this paper we propose to use timed automata (TAs) as event-triggered models for real time applications. A real time application may be a plant or simply a real time environment that generates sequences of events according to the pattern and timing constraints specified by a timed automaton. The sequences of events are picked up and served by a digital controller. We are aiming at synthesizing executable code (implementing a digital controller) for time-triggered architectures from such event-triggered specifications. As a first step, we abstract away from the computation tasks associated with the events, and only focus on the mechanism for admitting the events into the time-triggered environment before their observability deadlines expire. We have developed an automaton model called time-triggered automata (TTAs) to model finite state implementations of a controller that services the request patterns modeled by a timed automaton. A time-triggered automaton is deterministically driven by a timetable defined w.r.t. a fixed granularity of time. It is intended to be a finite state abstraction of computations realized on a time-triggered architectures [KB01,Kop98].

To relate the behaviors specified using TAs and TTAs, we have proposed a new semantics for TAs based on the non-instant observability of events, which gives rise to a simple notion of digitalization of timed languages. This enables to formulate the correctness criterion on TTA implementations of TA specifications as a language inclusion property. Our main result is that we can effectively decide whether a TTA correctly services the request patterns generated by a TA, that is, the TTA implements the TA specification.

We hope that the results presented in this paper may serve as a semantic basis for automatic code generation from abstract specifications in real time settings. Currently we are looking at how to automatically synthesize time-triggered automata from timed automata according to the correctness criterion presented in this paper.

References

[AD94] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[AFP+02] T. Amnell, E. Fersman, P. Pettersson, H. Sun, and W. Yi. Code synthesis for timed automata. *Nordic Journal of Computing*, 9(4):269–300, 2002.

[BCP+01] V. Bertin, E. Closse, M. Poize, J. Pulou, J. Sifakis, P. Venier, D. Weil, and S. Yovine. Taxys = Esterel + Kronos. A tool for verifying real-time properties of embedded systems. In *Proceedings of International Conference on Decision and Control, CDC'01*. IEEE Control Systems Society, 2001.

[BDM+98] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. Kronos: A model-checking tool for real-time systems. In *Proceedings of CAV'98*, volume 1427 of *LNCS*, pages 546–550. Springer–Verlag, 1998.

[BGK+96] J. Bengtsson, W. O. D. Griffioen, K. J. Kristoffersen, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. Verification of an audio protocol with bus collision using UPPAAL. In *Proceedings of CAV'96*, pages 244–256. LNCS, 1996.

[BPS00] Valérie Bertin, Michel Poize, and Joseph Sifakis. Towards validated real-time software. In *Proceedings of the 12 th Euromicro Conference on Real Time Systems*, pages 157–164. IEEE, 2000.

[CPP+01] E. Closse, M. Poize, J. Pulou, J. Sifakis, P. Venier, D. Weil, and S. Yovine. Taxys: a tool for the developpment and verification real-time embedded systems. In *Proceedings of CAV'01*, volume 2102 of *LNCS*. Springer–Verlag, 2001.

[DY00] A. David and W. Yi. Modeling and analysis of a commercial field bus protocol. In *Proceedings of Euromicro Conference on Real-Time*. IEEE Computer Society, 2000.

[FPY02] E. Fersman, P. Pettersson, and W. Yi. Timed automata with asynchronous processes: Schedulability and decidability. In *Proceedings of TACAS'02*, volume 2280 of *LNCS*, pages 67–82. Springer–Verlag, 2002.

[GHJ97] V. Gupta, T. Henzinger, and R. Jagadeesan. Robust timed automata. In *Hybrid and Real-Time Systems*, pages 331–345, Grenoble, France, 1997. Springer Verlag, LNCS 1201.

[HKH03] T.A. Henzinger, C.M. Kirsch, and B. Horowitz. Giotto: A time-triggered language for embedded programming. *Proceedings of the IEEE*, 91(1):84–99, January 2003.

[HMP92] T. Henzinger, Z. Manna, and A. Pnueli. What good are digital clocks? In *Proceedings of ICALP'92*, volume 623 of *LNCS*, pages 545–558. Springer-Verlag, 1992.

[KB01] H. Kopetz and G. Bauer. The time-triggered architecture. *Proceedings of the IEEE, Special Issue on Modeling and Design of Embedded Software*, 2001.

[Kop98] H. Kopetz. The time-triggered model of computation. *Proceedings of the 19th IEEE Systems Symposium (RTSS'98)*, 1998.

[KY04] Pavel Krčál and Wang Yi. Decidable and undecidable problems in schedulability analysis using timed automata. In *Proceedings of TACAS'04*, volume 2988 of *LNCS*, pages 236–250. Springer–Verlag, 2004.

[LPY97] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a Nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1(1–2):134–152, October 1997.

[LY97] K. G. Larsen and W. Yi. Time-abstracted bisimulation: Implicit specifications and decidability. *Information and Computation*, 134(2):75–101, 1997.

[OW03] J. Ouaknine and J. Worrell. Revisiting digitization, robustness and decidability for timed automata. In *Proceedings of LICS'03*, pages 198–207. IEEE Press, 2003.

[STY03] J. Sifakis, S. Tripakis, and S. Yovine. Building models of real-time systems from application software. *Proceedings of the IEEE, Special issue on modeling and design of embedded systems*, 91(1):100–111, 2003.

[WDR04] M. De Wulf, L. Doyen, and J.-F. Raskin. Almost ASAP semantics: From timed models to timed implementations. In *Proceedings of HSCC'04*, volume 2993 of *Lecture Notes in Computer Science*, pages 296–310. Springer–Verlag, 2004.

[WH04] L. Waszniowski and Z. Hanzálek. Analysis of real time operating system based applications. In *Proceedings of FORMATS'03*, volume 2791 of *LNCS*, pages 219–233. Springer-Verlag, 2004.