A Conservative Approximation Method for the Verification of Preemptive Scheduling using Timed Automata

Gabor Madl
University of California, Irvine
Irvine, CA 92697, USA
gabe@ics.uci.edu

Nikil Dutt
University of California, Irvine
Irvine, CA 92697, USA
dutt@ics.uci.edu

Sherif Abdelwahed
Mississippi State University
Starkville, MS 39762, USA
sherif@ece.msstate.edu

Abstract

This paper presents a conservative approximation method for the real-time verification of asynchronous event-driven distributed systems. This problem is known to be undecidable in the generic setting. The proposed approach is based on composable timed automata models that provide a sufficient condition to determine schedulability. We demonstrate the method on a real-time CORBA avionics design.

1. Introduction

Asynchronous event-driven communication is widely used in modern distributed real-time embedded (DRE) systems. Reducing synchronizations in event-driven systems can simplify implementation, prevent blocking waits, reduce energy consumption, and provide better throughput and flexibility. Providing formal guarantees on real-time properties in asynchronous event-driven systems, however, remains a key challenge.

Stopwatch automata [1] were proposed as a model of computation that can express preemptable tasks in asynchronous event-driven systems. It was shown that reachability analysis on the composition of stopwatch automata as task graphs (integration graphs) is undecidable [2], [3] if the following conditions are met: (1) tasks use event-based asynchronous triggering (i.e. a target task starts whenever its source finishes) on a distributed platform, (2) execution times are specified as continuous-time intervals, (3) preemptions may occur anytime within the continuous-time execution interval. In this paper, we refer to systems that satisfy these three conditions as *preemptive event-driven asynchronous real-time systems with execution intervals (PEARSE)*. PEARSE are a subset of DRE systems, that do not require global synchronization.

This paper presents a conservative approximation method for the verification of PEARSE models, using timed automata [4] model checking methods. The reachability problem is decidable on timed automata, therefore we provide an implementable method for the automatic real-time verification of PEARSE models.

The proposed method approximates each stopwatch automaton (S) using an approximate timed automaton (T). We show that the stopwatch automaton accepts all the time traces that the timed automaton accepts by showing that the language that T accepts is a subset of the language that S accepts $(L(T) \subseteq L(S))$. This problem is known as the language inclusion problem [5]. Since $L(T) \subseteq L(S)$ holds, there are no timed traces that the timed automaton accepts, but the stopwatch automaton does not accept, therefore the approximation is conservative. Accordingly, the proposed analysis provides a sufficient condition to determine the schedulability of preemptive event-driven asynchronous real-time systems with execution intervals (PEARSE).

The remainder of this paper is organized as follows. Section 2 describes related work; Section 3 reviews the background for real-time systems' analysis; Section 4 describes the problem statement; Section 5 presents the proposed method for the verification of preemptive scheduling, and proves the conservative nature of the approximation; Section 6 demonstrates the approach on a real-time CORBA application and Section 7 presents concluding remarks.

2. Related Work

The reachability problem on the composition of stopwatch automata as a task graph is undecidable in general, since it can be mapped to the halting problem [2]. The schedulability of preemptive multi-processor systems is undecidable using timed automata in the generic case [3], as timed automata cannot directly model stopwatches. Therefore, DRE system designers typically restrict the simultaneous occurrence of the three conditions used to define PEARSE systems to allow real-time analysis.

(1) Restricting asynchronous triggering: Time-triggered approaches [6] are becoming common in mission-critical systems. Classic schedulability analysis methods provide sufficient conditions for schedulability [7], [8] in predictable periodic real-time systems, but they do not address the dynamics of events, race conditions, and the non-deterministic execution order of tasks. These methods are typically overly conservative or unapplicable for PEARSE.

A generic periodic task and scheduling model for preemptive systems based on timed automata was introduced in [9]. This work – like the method proposed in this paper – uses the idea of discretizing timed automata clocks for the modeling of preemptive systems, but restricts asynchronous triggering, and is therefore only applicable to periodic task models. The models described also do not capture execution intervals and therefore cannot detect non-WCET deadline misses as shown in Figure 3, and may provide false positives in asynchronous event-driven systems.

(2) Restricting continuous-time execution intervals: Synchronous languages [10] propose a common mathematical model for synchronous systems with deterministic concurrency, but do not address non-deterministic execution times, arrival times, and asynchronous event-triggering typical in PEARSE systems. In Giotto [11], deterministic execution times are enforced to facilitate formal analysis, based on the concept of logical time. A method based on stopwatch automata [1] was proposed for the job-shop scheduling of preemptive systems [12]. This method is applicable for the real-time verification of PEARSE models, where execution times are given as constants. This restriction turns the reachability analysis decidable, as a single simulation trace can verify this model. However, this approach is too coarse for practical analysis, as execution times are often non-deterministic in DRE systems.

(3) Restricting preemptions: A method to analyze non-preemptive scheduling based on timed automata [4] on single processor architectures was proposed in [13]. A timed automata-based approach for the thread-level analysis of DRE systems is presented in [14]. We have presented a formal method for the real-time verification of non-preemptive DRE systems based on timed automata in [15], and for performance estimation based on discrete event simulations in [16]. Our earlier approach for modeling preemptive systems using timed automata [17] restricts preemptions to occur at discrete time steps.

The three restrictions described above involve cost, performance, and energy consumption overheads, as deterministic behavior has to be enforced on a distributed platform. These costs can often be justified only in the context of mission-critical systems. In most DRE systems, including consumer electronics, PEARSE models are commonly used. The industry practice for the analysis of PEARSE consists mostly of directed testing and random simulations. Although this approach may be helpful, it can only show the presence of timing violations, not their absence.

In this paper we propose a conservative approximation method that allows the simultaneous use of (1) asynchronous event-based triggering, (2) continuous-time execution intervals, and (3) preemptions that may happen anytime within a task's execution interval. To the best of our knowledge, the only alternative for the real-time analysis of systems that use the above three conditions (PEARSE systems) is stopwatch (hybrid) automata model checking. Preliminary experiments for approximating stopwatch reachability analysis using timed automata were described in [18], but the method is not guaranteed to terminate, since reachability analysis is undecidable on stopwatch automata in general.

3. Background

Given a finite set of clocks C a *valuation* for the clocks is a function $v : C \rightarrow \mathbb{R}_{\geq 0}$ that assigns a value for each clock from the domain of non-negative real numbers. The valuation of clock $c_i \in C$ is denoted v_i. $\mathcal{B}(\mathcal{C})$ is the set of *clock guards* γ, that are of the form $c_i \diamond \mathbb{N}$, $c_i - c_j \diamond \mathbb{N}$, $\diamond \in \{=, <, >, \leq, \geq\}$. A valuation v satisfies clock guard γ, if for all expressions in γ $v_i \diamond \mathbb{N}$, $v_i - v_j \diamond \mathbb{N}$ is satisfied, respectively. Time progress is captured as clock c_t with a constant rate of 1 ($\dot{v_t} = 1$). The initial value of c_t is denoted v_{t_0}, and $v_{t_0} = 0$.

Definition 1: A stopwatch automaton is a tuple $SA = (L, l_0, C, E, \mathit{Act}, \mathit{Inv})$ consisting of the following components:

- a finite set L of vertices called *locations*;
- the initial location $l_0 \in L$;
- a finite set C of real-valued clocks;
- a finite set of edges $E \subseteq L \times \mathcal{B}(\mathcal{C}) \times \mathit{Act} \times 2^C \times L$ called transitions, where $\langle l, \gamma, \alpha, \lambda, l' \rangle \in E$;
- a labeling function $\mathit{Act} : L \times C \rightarrow \{0, 1\}$, that defines the rates of clocks ($c_i \in C$) in locations as differential functions $\dot{v_i} = k_l$, where k_l is a constant 0 or 1 in each location;
- a labeling function $Inv : L \rightarrow \mathcal{B}(\mathcal{C})$, that is called the invariant of the location.

Definition 2: A state of the stopwatch automaton is defined as a pair (l, v) where $l \in L$ and v is the valuation of the clocks in C. The set of states is denoted S.

Definition 3: The *semantics* of a stopwatch automaton $SA = (L, l_0, C, E, \mathit{Act}, \mathit{Inv})$ is given as a transition system $T_{SA} = (S, s_0, \rightarrow)$ where S is the set of states, s_0 is the initial state, and the *step relation* $\rightarrow$ is the union of the *jump* (discrete) transitions:

- $(l, v) \xrightarrow{j} (l', v')$ if $\exists \langle l, \gamma, \alpha, \lambda, l' \rangle \in E$ such that γ and $Inv(l')$ are satisfied and v' = α,

and *flow* (continuous) transitions:

- $(l, v) \xrightarrow{f} (l, v')$ such that for each $v'_i \in v'$, $v'_i = v_i + x \cdot \mathit{Act}(l, c_i)$, where $x \in \mathbb{R}_+$ and $\mathit{Act}(l, c_i)$ is the labeling function that defines the rate of clock c_i as either 0 or 1 as introduced in Definition 1 ($\dot{v_i} \in \{0, 1\}$).

A *run* of the SA is a finite or infinite sequence of alternating discrete and continuous transitions of T_{SA}: $\rho : s_0 \xrightarrow{j} s_1 \xrightarrow{f} s_2 \xrightarrow{j} s_3 \ldots$ or $\rho : s_0 \xrightarrow{f} s_1 \xrightarrow{j} s_2 \xrightarrow{f} s_3 \ldots$

Definition 4: A timed automaton is a subclass of stopwatch automaton, where the rates of clocks are set to constant 1 $(\forall l \in L)(\forall c \in C)\mathit{Act}(l, c) = 1$.

4. Problem Formulation

4.1. Stopwatch as a Model for a Preemptable Real-time Task

A stopwatch is a clock that can be reseted, stopped, and resumed, providing a simple model for a preemptable real-time task. The execution time of a task can be represented as a stopwatch as shown in Figure 1. Time is represented as clock c_t, and the stopwatch clock is c_{sw}. The valuation of these clocks is v_t and v_{sw} as defined in Section 3.

The stopwatch makes a transition to the stop location from its initial (idle) location when it receives an enable_i event that signals that the task is ready for execution. The ? sign after an event denotes an input (receive) event, and the ! sign after an event denotes an output (send) event as used in [19]. Whenever the task is scheduled for execution, the stopwatch makes a transition to the run location, and whenever the task is preempted, the stopwatch moves to the stop location. v_{sw} represents the valuation of the stopwatch clock. We refer to the stopwatch shown in Figure 1 as *Task Stopwatch Automaton (TSA)* in this paper. When the task finishes its execution, the TSA moves to the finish location. We say that the TSA *executes* iff it is in the run location, and it is *preempted* iff it is in the stop location, and we refer to the time spent in the run location as *execution time*. Figure 1 can be extended to model periodic tasks by adding a transition to the idle location from the run location instead of the finish location. In this paper we use the simple one-time executing task shown in Figure 1 for simplicity.

A task may have a *best case execution time* (bcet), that corresponds to the shortest, and a *worst case execution time* (wcet), that corresponds to the longest time in which the task may finish its execution. Time for execution is counted from the time of the enable_i event (v_{t_0}), and does not include time spent in the stop location. The following constraints are implied by the definitions of the stopwatch model, best case and worst case times:

$$0 \leq v_{sw} \leq v_t,\ 0 \leq v_{sw} \leq \texttt{wcet},\ 0 \leq \texttt{bcet} \leq \texttt{wcet} \quad (1)$$

Deadlines, denoted dl for a given task, are constraints on the maximum time from the time of the enable_i event to the time when the automaton makes a transition to the finish location.

Definition 5: A real-time task is schedulable if it always finishes its execution before its respective deadline. A task is then schedulable iff $v_t \leq \texttt{dl}$ when $v_{sw} = \texttt{wcet}$.

The alphabet of the TSA is $\Sigma = \{$start, stop, enable_i, $\text{enable}_\text{j}\}$. The start and stop events are controlled by a (set of) scheduler(s), the task receives the enable_i event from its source task, and sends out the enable_j event when it finishes its execution. See Section 4.2 for the formal definition of composition rules.

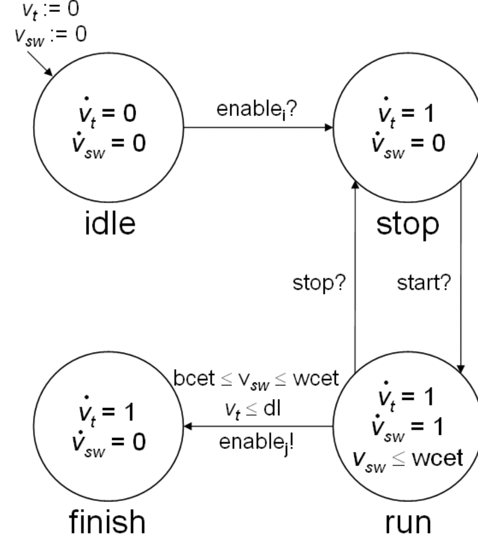

Figure 1. Task Stopwatch Automaton (TSA) – Model of a Preemptable Real-time Task

A *timed word* is of the form $(\sigma_0, \tau_0)(\sigma_1, \tau_1) \ldots (\sigma_n, \tau_n)$, where $\sigma_0, \sigma_1, \ldots, \sigma_n \in \Sigma$ denote events, and $\tau_1, \tau_2, \ldots, \tau_n \in \mathbb{R}_{\geq 0}$ denote the timestamps of events. The set of timed words is the *timed language* on which the TSA operates. We can express the syntax of the (untimed) language that the TSA accepts using the following regular expression:

$$S_{L(S)} = \text{enable}_\text{i}\ \text{start}\ (\text{stop start})^*\ \text{enable}_\text{j} \quad (2)$$

The timestamps of all events have to be less than dl in a timed word in order for the TSA to accept the word. We denote the timestamps of events in the $[0 \ldots \texttt{dl}]$ interval as $\tau_1, \tau_2, \ldots, \tau_e$, where τ_1 denotes the enable_i event, and t_e denotes the last start event. Note that e is always an even number according to Equation 2. The TSA accepts the timed language $L(S)$ as described in Equation 2, and satisfies the following constraint:

$$\sum_{i=1}^{\frac{e}{2}} \tau_{2i} - \tau_{2i-1} \leq \texttt{dl} - \texttt{wcet} \quad (3)$$

Equation 3 states that a task is schedulable if it spends at most dl - wcet time in the stop location in the TSA within the $[0, \texttt{dl}]$ interval for clock c_t, since in this case it can execute for wcet time before its deadline. Figure 2 shows the constraints implied on the TSA clock for a schedulable task. v_{sw} is in the $[0, \texttt{wcet}]$ domain, v_t is in the $[0, \texttt{dl}]$ domain, and the slope of v_{sw} is at most 1 ($\dot{v}_{sw} \in \{0, 1\}$). The valid clock assignments define a parallelogram, and all other assignments result in possible deadline violations. Note that since this model is an initialized stopwatch automaton, reachability is decidable [20], therefore we can verify the

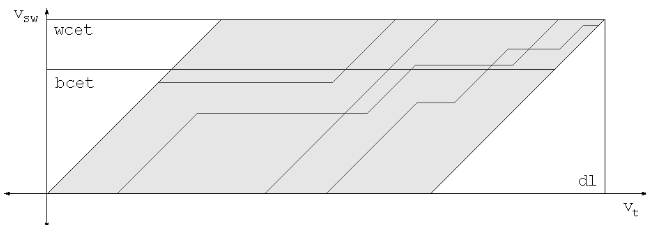

Figure 2. Clock Constraints on Stopwatches for Schedulability

schedulability of a single preemptable task, given that we know when start/stop events occur. The darker lines show a few stopwatch clock valuation traces that model the execution of schedulable real-time tasks.

4.2. Composable Stopwatch Automata as a Model for PEARSE

In Section 4.1 we described how TSA can model a preemptable real-time task. In this section we consider how the composition of stopwatches can represent PEARSE. In this paper we represent PEARSE as task graphs $G_S = (X_S, E_S)$, where the set of vertices represent real-time tasks modeled as stopwatches, and edges represent dependencies between the tasks $E_S \subseteq X_S \times X_S$. A vertex with no incoming edge(s) is a *source*, and a vertex with no outgoing edge(s) is a *terminal*. This task graph model is a subclass of *integration graphs* defined in [2]. In our model, each hybrid automaton in the integration graph is a TSA.

There are two ways in which TSA models compose in G_S; serial and parallel composition. When *parallel composition* is used, the composed automata operate independently from each other. In graph G_S, two TSA compose using parallel composition, if none of them is reachable from the other on a directed path. We denote parallel composition between two vertices $x_i, x_j \in X_S$ as $x_i \oplus x_j$. The $\oplus$ operator is associative, distributive and commutative.

Assume that there is an edge in graph G_S between two vertices $(x_i, x_j) \in E_S$, so task x_j depends on task x_i. Then the enable_j transition in TSA$_i$, and the enable_i transition in TSA$_j$ can only be taken simultaneously. We refer to this case as *serial composition*, and define it as follows. Denote the language of x_i as $L(S)_i$, and the language of x_j as $L(S)_j$. Denote the alphabet of x_i as $\Sigma_i = \{\text{start}_\text{i}, \text{stop}_\text{i}, \text{enable}_{\text{i}_\text{i}}, \text{enable}_{\text{j}_\text{i}}\}$, and the alphabet of x_j as $\Sigma_j = \{\text{start}_\text{j}, \text{stop}_\text{j}, \text{enable}_{\text{i}_\text{j}}, \text{enable}_{\text{j}_\text{j}}\}$. By definition, the serial composition of x_i and x_j means that the timestamp of $\text{enable}_{\text{j}_\text{i}}$, and the timestamp of $\text{enable}_{\text{i}_\text{j}}$ is the same. We denote serial composition between two vertices $x_i, x_j \in V_S$ as $x_i \otimes x_j$. The $\otimes$ operator is associative, distributive, but not commutative, since if x_i depends on x_j is not the same case as when x_j depends on x_i.

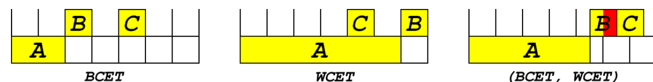

Figure 3. Motivating Example for a Non-WCET Deadline Miss

For the sake of simplicity we do not consider buffers or communication delays between tasks. Further, without losing generality, we disallow multiple sources to tasks $(\forall(x_a, x_b, x_j \in V_S)((x_a, x_j) \in E_S \wedge (x_b, x_j) \in E_S) \rightarrow x_a = x_b)$, each task may depend on at most one task directly. We do allow multiple dependents for tasks, as these can be modeled as synchronizations between transitions, and do not require buffers.

We point out that this model can be easily extended to include FIFO channels modeled as timed automata, that can express many-to-many connections, and communication delays as well. Moreover, the model can be extended to include periodic tasks modeled as timed automata that broadcast enable_i events with some rate. The start and stop events may be controlled by a (set of) scheduler(s), providing a model of computation that can express PEARSE in a formal setting. Please see [15] for an approach to model real-time CORBA applications using timed automata as a model of computation for real-time analysis.

4.3. Problem Description

Integration graphs may be used to compose stopwatches using events, to express PEARSE as a network of stopwatch models using preemptive scheduling. Although the reachability analysis of a single initialized stopwatch automaton is decidable [20], reachability analysis on integration graphs is undecidable in general, more specifically if the conditions defining PEARSE are met: (1) tasks use event-based asynchronous triggering (i.e. a target task starts whenever its source finishes) on a distributed platform, (2) execution times are specified as continuous-time intervals, (3) preemptions may occur anytime within the continuous-time execution interval [2], [3].

In this paper we propose a conservative approximation method for the verification of preemptive scheduling in PEARSE designs. We approximate stopwatch automata using timed automata by discretizing clocks, to "store" time passed before a preemption. The practical benefit of this method is that we provide a decidable technique for the real-time verification of PEARSE.

In event-driven systems it is not enough to consider the worst case times of tasks in general. Consider the simple example shown in Figure 3. Task A is running on `machine_1`, and tasks B and C are running on `machine_2`. Task A starts at time 0, and may finish its execution time anytime within the $[2, 6]$ interval. Task B

starts its execution whenever task A finishes its execution, and executes for 1 time unit. Task C starts its execution at time 4, and executes for 1 time unit. We assume that task B has higher priority than task C, and that the deadlines for task B and task C are 1.2 time units. The system is schedulable when task A executes for its bcet time as shown in the left of Figure 3, and it is schedulable when task A executes for its wcet time as shown in the middle of Figure 3. However, if task A finishes its execution at time 5.5, task C will miss its deadline as it has to wait for task B. Therefore, the analysis has to capture execution intervals in continuous time, otherwise it may lead to false positives; unschedulable designs that cannot be detected at design time.

We achieve this goal by mapping preemptive scheduling to non-preemptive scheduling. Timed automata can express non-preemptive scheduling with execution intervals [13], [15], and reachability analysis is decidable on timed automata [4].

5. Conservative Approximation of Integration Graphs

In this section we describe the conservative approximation method for the reachability analysis of integration graphs. We implement this approximation in two steps. In the first step, we map each TSA in graph G_S (defined in Section 4.2) to a timed automaton. We refer to this timed automaton as *Task Timed Automaton (TTA)* (described in detail in Section 5.1). In the second step we consider how approximation errors can be considered in the analysis of task graphs. We denote the language that the TTA accepts as $L(T)$. Then we show that $L(T) \subseteq L(S)$, that implies the conservative nature of the approximation.

5.1. Mapping the TSA to TTA

In this section we show how the TSA can be mapped to the TTA. We represent preemptable tasks as TSA as shown in Figure 1. We introduce a generic timed automaton template for preemptable tasks as shown in Figure 4. The locations denoted as $\mathsf{run}_{\mathsf{x,y}}$ in Figure4 represent the run location of the TSA, the $\mathsf{stop}_{\mathsf{x,y}}$ locations represent locations where the task is preempted (location stop in Figure 1). The x index represents discrete checkpoints denoting time passed, and y represents the number of preemptions encountered during the run of the TTA. We say that the TTA *executes* iff it is in a $\mathsf{run}_{\mathsf{x,y}}$ location, and it is *preempted* iff it is in a $\mathsf{stop}_{\mathsf{x,y}}$ location, and we refer to the time spent in $\mathsf{run}_{\mathsf{x,y}}$ locations as *execution time*.

We denote the time unit used for the discretization as t_u. We partition the wcet time of a task to n equal-size intervals, and a smaller interval representing the fraction of the division of wcet by t_u. If t_u is a divisor of wcet, then the locations $\mathsf{run}_{\mathsf{n,y}}$ representing the fractions are not required, and the value of n and m needs to be decreased by 1 in all guards in Figure 4. We define the constants used in the template as follows:

$$n = \lfloor \frac{\texttt{wcet}}{t_u} \rfloor, 1 \leq m \leq \frac{\texttt{dl}}{t_u}, k = \lfloor \frac{\texttt{bcet}}{t_u} \rfloor \qquad (4)$$

Key restriction: *Since the TTA is an approximation of the TSA, it can only express bounded numbers of preemptions (per task).* Note that this bound *does not restrict periodic execution.* If the execution of the task graph is repeatedly triggered by a periodic (or aperiodic) event, the task can get preempted limited times during a single execution of the task graph. However, there is no bound on the number of executions, and thus the overall number of preemptions.

The constant m represents the maximum number of preemptions that the TTA can capture. This is a weak restriction, as in practical systems tasks are not preempted infinitely often. In fact, frequent preemptions can significantly degrade the system performance and therefore should be avoided by design. Moreover, since task graphs (and dependencies between tasks) are fixed, one can calculate how many times a task may get preempted during a single execution of the task graph. *The value of m is not affected by how many times the task graph is executed, or whether the design is periodic.* The value of m is proportional to $\frac{\texttt{dl}}{t_u}$, and the maximum value of m is $\frac{\texttt{dl}}{t_u}$. We prove this statement at the end of Section 5.2, where we describe the timed language that the TTA accepts.

Each $\mathsf{run}_{\mathsf{x,y}}$ location represents t_u time spent executing, due to the invariants $v_{ta} \leq t_u$ (see Definition 1). Since the difference between the wcet and bcet time of a task may be larger than the time unit t_u used for the discretization, we may need to introduce transitions from several $\mathsf{run}_{\mathsf{x,y}}$, $x \in \{0 \ldots n-1\}, y \in \{0 \ldots m\}$ locations, to provide a way for the automaton to jump to the finish location (shown as curvy arrows in Figure 4). We introduce a transition from each $\mathsf{run}_{\mathsf{x,y}}$, $k \leq x + y$ location to the finish location, where the constant k from Equation 4 is used to calculate the indeces of $\mathsf{run}_{\mathsf{x,y}}$ locations from which the finish location is reachable, as the example shows in Figure 4.

The model shown in Figure 4 does not use any variables other than the valuation of a single clock (v_{ta}), and the valuation of the global time clock (v_t). The constants m, n, k used in the guards can be computed before the verification process, and therefore do not require extensions to timed automata [4]. However, there exist some extensions to timed automata that allow the use of integer variables, and the value of k, n, m can be encoded in integer variables, and therefore pre-computing these constants is not required when using modern model checkers such as UPPAAL [19] or the Verimag IF toolset [21].

Figure 4. Task Timed Automaton (TTA) – Approximating a Preemptable Real-time Task

5.2. Analysis of the Timed Automaton Approximation

In Section 5.1 we presented the construction rules for the approximation of the TSA using the TTA. In this section we prove the conservative nature of the approximation using the language inclusion problem.

The alphabet of the TTA is the alphabet of the TSA, Σ = {start, stop, enable$_i$, enable$_j$}. However, the TTA can only capture a bounded number of preemptions (per task). The syntax of the (untimed) TTA language can be described using regular expressions as follows:

$$S_{L(T)} = \text{enable}_\text{i}\ \text{start}\ (\text{stop start})^{0\ldots m}\ \text{enable}_\text{j} \qquad (5)$$

We see that $S_{L(T)} \subseteq S_{L(S)}$. The run$_{x,y}$ locations of the TTA represent "checkpoints" in time; they store the discrete clock value and the number of preemptions occured during the execution of the task. If a task is in the run$_{a,b}$ location, then it has executed for at least $a \cdot t_u$ time, and has encountered b preemptions.

Definition 6: We define the valuation $v_x = v_{ta} + x$, where v_{ta} is the valuation of c_{ta} as in Figure 4, and x is the index of locations run$_{x,y}$ or stop$_{x,y}$, where the automaton resides. Valuation v_x has a discrete-time component, that stores the checkpoints that have already been encountered, and a continuous-time component, that measures the time between checkpoints. Since timed automaton clocks cannot be resumed like a stopwatch, the valuation v_x is only an approximation of v_{sw}, that models the actual execution time accurately. Note that v_x is not used in the TTA model directly, we define it for the sole purpose of measuring the imprecision of using v_x to approximate clock value v_{sw}. To establish the relationship between the TSA and the TTA, we compare them on timed words that follow the syntax of the (untimed) regular expression $S_{L(T)}$. We assume that both the TSA and the TTA receive events in the same order, and with the same timestamps.

Proposition 1: For any timed word r that follows the syntax of the (untimed) regular expression $S_{L(T)}$, during the co-simulation of word r on both the TSA and the TTA $v_{sw} - m \cdot t_u \leq v_x \leq v_{sw}$ holds from time 0 to the timestamp of the last event, where v_{sw} is the valuation of TSA clock c_{sw} shown in Figure 1, v_x is the valuation defined in Definition 6, and m is the number of stop events in the timed word (the number of preemptions).

Proof: Since both the TSA and the TTA receive events with the same timestamps, their transitions will happen simultaneously. Whenever the TSA receives a start event, it makes a transition to the run location, and whenever it receives a stop event it moves to the stop location. Similarly, whenever the TTA receives a start event, it makes a transition to one of the run$_{x,y}$ locations, and whenever it receives a stop event it moves to one of the stop$_{x,y}$ locations.

The stopwatch clock valuation v_{sw} increases with a slope of 1 between start and stop events in the run location, and is constant between stop and start events. The valuation v_x also increases with a slope of 1 between start and stop events in the run$_{x,y}$ locations, due to the continuous-value timed automaton clock valuation v_{ta} shown in Figure 4. Whenever a preemption happens, v_{ta} is reset (shown in Figure 4). Therefore, with each preemption, valuation v_x loses the value stored in the continuous-time component (v_{ta} becomes 0), whereas clock v_{sw} keeps its value, therefore, $v_x \leq v_{sw}$.

When the TTA receives a stop event, the clock valuation v_{ta} is in the $[0, t_u]$ interval, due to the invariants $v_{ta} \leq t_u$.

Since the clock is reset, v_x decreases by at most t_u time, while v_{sw} hold its value when a preemption occurs. If there are m preemptions, the clock value v_x may decrease by at most $m \cdot t_u$ time compared to clock value v_{sw}, therefore $v_{sw} - m \cdot t_u \leq v_x$. □

Proposition 1 is the key to quantify the imprecision of the timed automaton approximation. It shows that valuation v_x increases slower than – or in the case when no preemptions occur with the same slope as – clock value v_{sw} due to the fact that the TTA does not keep track of the time spent in the $\mathsf{run_{x,y}}$ location where it has received the stop event. These results imply that for the same timed word, it takes at least as much time for the valuation v_x to reach a given value, as it does for the stopwatch clock valuation v_{sw}.

Recall that wcet and bcet denote the longest and shortest possible execution times of a task, respectively, not including the time spent in the stop location. If we denote the time that the TSA has spent in the run location as τ_{run}, then $v_{sw} = \tau_{run}$, based on the fact that v_{sw} is simply a function of v_t.

In the TTA model we also use the wcet, bcet parameters to obtain guards on transitions. Proposition 1 shows that valuation v_x follows clock value v_{sw} with some imprecision. The time that the TTA spends in $\mathsf{run_{x,y}}$ locations equals τ_{run}, because anytime the TTA receives a start event it makes a transition to a $\mathsf{run_{x,y}}$ location, and anytime it receives a stop event it moves to a $\mathsf{stop_{x,y}}$ location, just like the TSA does. Therefore, according to Proposition 1, $v_{sw} - m \cdot t_u \leq v_x \leq v_{sw} = \tau_{run}$.

Definition 7: We refer to the valuation of global time clock (v_t) when $v_x = \mathtt{wcet}$ as actual worst case execution time, and denote it as t_{wcet}. The valuation of global time clock (v_t) when $v_x = \mathtt{bcet}$ is referred to as actual worst case execution time, and denoted as t_{wcet}

Proposition 2: For any timed word r that follows the syntax of the (untimed) regular expression $S_{L(T)}$, if $v_x = \mathtt{wcet}$ holds anytime during the run of word r on the TTA, then $\mathtt{wcet} \leq t_{wcet}$.

Proof: The invariants $v_{ta} \leq t_u$ in the TTA imply that the TTA spends at most t_u time in each $\mathsf{run_{x,y}}$ location. Time spent in $\mathsf{stop_{x,y}}$ locations is not part of the execution time and therefore does not contribute to t_{wcet} (only to the deadline). Moreover, we reset the continuous-time component (clock valuation v_{ta} in Figure 4) whenever we leave a $\mathsf{stop_{x,y}}$ location, therefore the value of clock valuation v_x when it leaves the stop location is less than or equal to its value when it entered the location. Neither t_{wcet}, nor v_x increases by passing through $\mathsf{stop_{x,y}}$ locations, and therefore we can abstract these location out here. If the TTA receives no stop events, it spends t_u time in n locations, then $\mathtt{wcet} - n \cdot t_u$ time in the last location, therefore if no preemptions occur, t_{wcet} is wcet in the TTA.

Create a directed graph $G_M = (V_M, E_M)$ such, that for each $\mathsf{run_{x,y}}$ location add a vertex $v_{x,y}$ in graph G_M. For all vertices $v_{x,y}, x \in 0 \dots n-1, y \in 0 \dots m-1$ add a directed edge from $v_{x,y}$ to $v_{x+1,y}$, and a directed edge from $v_{x,y}$ to $v_{x,y+1}$. Add a terminal vertex z to graph G_M, and add edges from each $v_{n,y}$ to z. We only add edges from $v_{n,y}$ locations because we are interested in the worst case execution time of the task, and therefore we require the automaton to go through (at least) n $\mathsf{run_{x,y}}$ locations. The edges in G_M specify the order in which $\mathsf{run_{x,y}}$ locations can follow each other in the TTA, the source represents the initial idle location, and the terminal represents the finish location. The graph G_M is an abstract model of the TTA shown in Figure 4. The shortest path in G_M from $v_{0,0}$ to t is the $s, v_{0,0}, v_{1,0}, \dots v_{n,0}, t$ path, that corresponds to the case when the TTA receives no stop events. Anytime the TTA receives a stop event, it needs to go through additional $\mathsf{run_{x,y}}$ locations, and therefore t_{wcet} increases. □

Proposition 3: For any timed word r that follows the syntax of the (untimed) regular expression $S_{L(T)}$, if $v_x = \mathtt{bcet}$ holds anytime during the run of word r on the TTA, then $t_{bcet} \leq \mathtt{bcet}$.

Proof: We build on the G_M graph introduced in Proposition 2. Since we are interested in the best case, we add an edge from each $v_{x,y}$ vertex to t, where $k \leq x + y$, not just from $v_{n,y}$ vertices. Since we introduced edges from each $\mathsf{run_{x,y}}$, $k \leq x+y$ vertex, the shortest path from s to t is $k+2$ $(k \cdot v_{x,y}$ vertices, plus start (s) and terminal (t)locations), regardless of the value of y, that represents the number of preemptions. Each vertex represents a $\mathsf{run_{x,y}}$ location, where we spend at most t_u time, due to the invariant $v_{ta} \leq t_u$, and therefore v_t is at most $k \cdot t_u$ when the TTA reaches a location that has a transition to the finish location. The guard on all transitions from $\mathsf{run_{x,y}}$ locations to the finish location is $\mathtt{bcet} - k \cdot t_u$, and $\mathtt{bcet} - k \cdot t_u + k \cdot t_u = \mathtt{bcet}$, therefore t_{bcet} is bcet or less in the TTA. □

Now that we established the relation between the TSA and the TTA, we focus on the language that the TTA accepts. The timestamps of all events have to be less than dl in a timed word in order for the TSA to accept the word. We discard the $\mathsf{enable_j}$ event, since it might not correspond to the worst case. Similarly to the notations used with the TSA, we denote the timestamps of events in the $[0 \dots \mathtt{dl}]$ interval as $t_1, t_2, \dots, t_e$, where t_1 denotes the $\mathsf{enable_i}$ event, and t_e denotes the last start event.

The number of preemptions $m = \frac{e}{2}$ in Proposition 1, since every second event is a stop event (and e is even). Therefore, $\sum_{i=1}^{\frac{e}{2}} t_u = m \cdot t_u$, corresponding to the maximum difference between clocks c_{sw} and v_x on any timed word over Σ, as shown in Proposition 1. Accordingly, we conclude, that the TTA accepts the timed language that has a syntax as described in Equation 5, and satisfies the following constraint:

$$\sum_{i=1}^{\frac{e}{2}} \tau_{2i} - \tau_{2i-1} + t_u \leq \mathtt{dl} - \mathtt{wcet} \qquad (6)$$

Equation 6 states that a task is schedulable if it spends at most $\mathtt{dl} - \mathtt{wcet} - m \cdot t_u$ time in $\mathsf{stop_{x,y}}$ locations in the TTA within the $[0, \mathtt{dl}]$ interval, since in this case it can execute for wcet time before its deadline. We need to subtract $m \cdot t_u$ from the available time to compensate for the imprecision of the timed automata approximation described in Proposition 1.

We now show that the maximum value of m is $\frac{\mathtt{dl}}{t_u}$ in Equation 4. From Equation 6 we see that the maximum value for m defined in Equation 4 is $\frac{\mathtt{dl}}{t_u}$, since the number of preemptions $m = \frac{e}{2}$, therefore in this case $\sum_{i=1}^{\frac{\mathtt{dl}}{t_u}} t_u = \mathtt{dl} \geq \mathtt{dl} - \mathtt{wcet}$, therefore the constraint specified in Equation 6 will never be satisfied when the number of preemptions encountered is more than $\frac{\mathtt{dl}}{t_u}$.

5.3. Language Inclusion Problem for a Single TTA/TSA Pair

The language inclusion problem for stopwatch automata can be described as follows; given two stopwatch automata T and S, are all timed traces accepted by T also accepted by S? For the proof we proceed with the common method of complementation and emptiness checking of the intersection [5]: $L(T) \subseteq L(S)$ iff $L(T) \cap \overline{L(S)} = \emptyset$.

Theorem 1: The TSA accepts the timed language over Σ that the TTA accepts: $L(T) \subseteq L(S)$.

Proof: $S_{L(T)} \subseteq S_{L(S)}$, therefore it is sufficient to show that $L(T) \subseteq L(S)$ holds over timed words that can be expressed using the (untimed) syntax $S_{L(T)}$. Let t_{stop} denote the expression $\sum_{i=1}^{\frac{e}{2}} \tau_{2i} - \tau_{2i-1}$, the time that the TSA spends in the stop location. The timestamps of events are the same in the TTA and TSA traces, as we compare the timed languages $L(T)$ and $L(S)$ word by word. Then, the time constraints on the timed language $L(T) \cap \overline{L(S)}$ can be expressed as $t_{stop} + \sum_{i=1}^{\frac{e}{2}} t_u \leq \mathtt{dl} - \mathtt{wcet} \cap t_{stop} > \mathtt{dl} - \mathtt{wcet}$. Since $t_u \in \mathbb{R}_{\geq 0}$, therefore $0 \leq \sum_{i=1}^{\frac{e}{2}} t_u$. Since t_{stop} cannot be both smaller than or equal to dl - wcet and less than dl - wcet, therefore the intersection of $L(S)$ and $L(T)$ is the empty set, that implies that the TSA accepts the timed language over Σ that the TTA accepts, and $L(T) \subseteq L(S)$ holds. □

5.4. The Effects of Composing TTA on the Approximation

We now show that since t_{wcet} of the TTA is larger than t_{wcet} of its corresponding TSA, and t_{bcet} of the TTA is smaller than t_{bcet} of its corresponding TSA, therefore the composition of TTA models is a conservative approximation of the composition of TSA.

Theorem 1 shows that the TSA accepts the language that the TTA accepts ($L(T) \subseteq L(S)$). As described in Section 4, the composition of TSA as a task graph turns reachability analysis undecidable [2], which motivated our work to approximate TSA task graphs using TTA task graphs. In this Section we show that the composition of TTA as a task graph (denoted as G_S in Section 4.2) does not invalidate the results of Theorem 1. For the conservative approximation of TSA task graphs, we replace each TSA in the task graph with a TTA. We denote this graph as $G_T = (X_T, E_T)$, where each vertex in set X_T is a TTA. As each timed automaton is also a stopwatch automaton, TTA compose using events the same way as TSA do. Graphs G_S and G_T are a representation of applying the $\oplus$ parallel composition operator, and the $\otimes$ serial composition operator to TSA and TTA models, respectively. Therefore, we need to consider how these operators may influence the timestamps of events.

When parallel composition is used between two automata $(x_k \oplus x_l), x_k \in X_T, x_l \in X_T$, then the two automata do not depend on each other and can be analyzed independently. Therefore, the parallel composition of TSA and TTA models does not influence the timestamps of events.

We now consider the case when serial composition is used between two automata $(x_k \otimes x_l), x_k \in X_T, x_l \in X_T$. Denote the language of x_k as $L(T)_k$, and the language of x_l as $L(T)_l$. Denote the alphabet of x_k as $\Sigma_k = \{\mathsf{start_k}, \mathsf{stop_k}, \mathsf{enable_{k_k}}, \mathsf{enable_{l_k}}\}$, and the alphabet of x_l as $\Sigma_l = \{\mathsf{start_l}, \mathsf{stop_l}, \mathsf{enable_{k_l}}, \mathsf{enable_{l_l}}\}$. Since the timestamp of $\mathsf{enable_{l_k}}$, and the timestamp of $\mathsf{enable_{k_l}}$ is the same by definition, therefore the timestamps of events in Σ_l may be influenced by the timestamp of $\mathsf{enable_{l_k}}$.

The $\mathsf{enable_{l_k}}$ event signals the end of the execution of TTA_k, and is raised when v_x is within the $[\mathtt{bcet_k}, \mathtt{wcet_k}]$ interval. Therefore, the timestamp of $\mathsf{enable_{l_k}}$ is influenced by the imprecision between v_x and v_{sw} described in Proposition 1. Proposition 2 shows, that if $v_{x_k} = \mathtt{wcet_k}$ holds, then $\mathtt{wcet_k} \leq t_{wcet_k}$. Also, Proposition 3 shows, that if $v_{x_k} = \mathtt{bcet_k}$ holds, then $t_{bcet_k} \leq \mathtt{bcet_k}$. Therefore, if $v_{x_k} = \mathtt{wcet_k}$ holds for a TTA, then the timestamp of $\mathsf{enable_{l_k}}$ is in the $[t_{bcet_k}, t_{wcet_k}]$ real-valued interval, and $[\mathtt{bcet_k}, \mathtt{wcet_k}] \subseteq [t_{bcet_k}, t_{wcet_k}]$. This implies that TTA_k can generate all the timestamps for event $\mathsf{enable_{l_k}}$, that its corresponding TSA_k model can generate, if $v_{x_k} = \mathtt{wcet_k}$ can be satisfied. If it cannot, then the TTA will report the task as unschedulable (that may or may not be true). Therefore, the proposed approximation method provides a sufficient, but not necessary condition, in general, to determine the schedulability of TSA models composed using the $\oplus$ and $\otimes$ operators.

6. Practical Application

We applied the proposed conservative approximation method to analyze a PEARSE design shown in Figure 5, loosely based on a real-time CORBA avionics application [15]. Tasks represent software components, and are

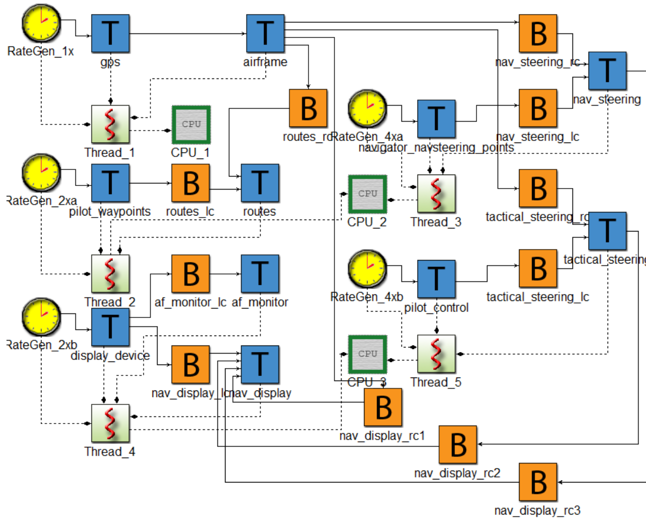

Figure 5. Real-time CORBA Avionics Application

Timer	Period
Rategen_1x	1000
Rategen_2xa	500
Rategen_2xb	500
Rategen_4xa	250
Rategen_4xb	250

Task	WCET	BCET	Deadline
gps	21	18	100
airframe	53	50	100
pilot_wayp...	37	35	300
routes	18	15	250
display_device	26	26	250
af_monitor	33	32	150
nav_display	14	12	150
nav_steering	69	65	150
navigator...	42	42	100
pilot_control	43	37	80
tactical_st...	58	52	100

Table 1. Parameters for the Real-time CORBA Case Study Shown in Figure 5

denoted as T, FIFO event channels are denoted B. Timers send out events periodically, driving the computation in the design. Arrows represent dependencies between tasks. Tasks are mapped to threads as defined by the dashed lines. Within each thread, fixed-priority non-preemptive scheduling is used, and fixed-priority preemptive scheduling is used between threads. Both CPU_2 and CPU_3 use preemptive scheduling. FIFOs are scheduled non-concurrently (i.e. they are always ready to execute). Communication between software tasks is fully asynchronous and event-driven. Overall, there are 11 tasks in the design and 11 FIFO buffers, that execute on 5 threads on 3 CPUs. Execution parameters for tasks are shown in Table 1.

Each task is represented as a TTA, and FIFOs are modeled as timed automata that buffer events. We can only illustrate the size and design of the model used for the analysis. Our approach for modeling FIFOs and scheduling policies is described in detail in [15]. We introduce an error location for each TTA to ensure that the model deadlocks whenever the deadline is exceeded. Thus, when no deadlocks occur then all deadlines are met.

We have checked the schedulability of the timed automata model using the the UPPAAL model checker [19] by issuing the A[] not deadlock macro (experiment 1). We then ran experiments where in each step we halved both the BCET and WCET of a single task (first gps, then airframe etc.) We used the highest possible precision for preemptive tasks, the value of the clock is saved at every integer value during the execution of a task. Experiments were executed on an Intel Core i7 i920 processor running at 4GHz, using 6GB three-channel RAM. Model checking time for the 12 experiments is shown in Figure 6, and the memory used is shown in Figure 7.

Both the verification time and memory consumption vary as a function of non-determinism, which is influenced by many factors, including the actual execution parameters, the size of execution intervals, the number of concurrently executing tasks, as well as the number of tasks. That said, complexity cannot simply be judged as a factor of size.

In experiment 3, where the WCET and BCET parameters for the airframe task are halved (to 27 and 25, respectively), the change results in a deadline miss in the nav_display task. This means that we did not find a sufficient condition for schedulability, and the design may or may not be schedulable. All other experiments proved the design schedulable with the given parameters. Decreasing the execution parameters for the tactical_steering task greatly increased verification time and memory consumption. The cause of the complexity increase is unknown to us, but we suspect that the changes have increased the non-determinism in the model (i.e. by introducing race conditions, or non-deterministic execution order).

Several improvements may increase scalability in real-life problems. First of all, the proposed method allows for *hierarchical model checking*, since preemptive components can be encapsulated into non-preemptive "wrappers", acting as a black box. Since intervals for communication are captured, there is no need to model all components at once. We plan to investigate this direction in the future. Second, UPPAAL does not take advantage of multi-core processors or distributed clusters. Model-checking algorithms that are CPU-bound rather than memory-bound – such as the algorithm described in [16] – have the potential to leverage multi-core hardware and may provide better performance in the future. There is room for optimization in current model checkers. Given the resources, the real-time verification of large-scale designs is within reach.

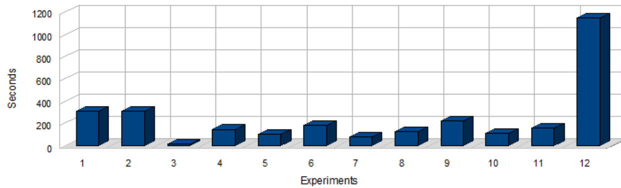

Figure 6. Model Checking Time

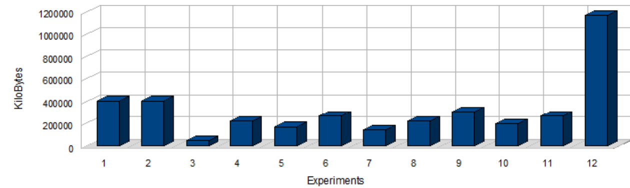

Figure 7. Model Checking Memory Consumption

7. Concluding remarks

This paper presents a conservative approximation method for the verification of preemptive event-driven asynchronous real-time systems with execution intervals (PEARSE). The proposed method is based on timed automata model checking methods, and inherently captures asynchrony and dependencies between tasks and provides a way for the formal analysis of practical embedded systems. We have shown that the approximation provides a sufficient, but not required condition to determine the schedulability of distributed asynchronous event-driven systems using preemptive scheduling. The practical application of the method was shown on a real-time CORBA avionics application.

References

[1] J. McManis and P. Varaiya, "Suspension Automata: A Decidable Class of Hybrid Automata," in *Proceedings of CAV*, 1994, pp. 105–117.

[2] Y. Kesten, A. Pnueli, J. Sifakis, and S. Yovine, "Decidable Integration Graphs," *Information and Computation*, vol. 150, no. 2, pp. 209–243, 1999.

[3] P. Krcal, M. Stigge, and W. Yi, "Multi-Processor Schedulability Analysis of Preemptive Real-Time Tasks with Variable Execution Times," in *Proceedings of FORMATS*, 2007, pp. 274–289.

[4] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, no. 2, pp. 183–235, 1994.

[5] J. E. Hopcroft, R. Motwani, Rotwani, and J. D. Ullman, *Introduction to Automata Theory, Languages and Computability*. Addison-Wesley Longman Publishing Co., Inc., 2000.

[6] H. Kopetz and G. Bauer, "The Time-Triggered Architecture," *Proceedings of the IEEE, Special Issue on Modeling and Design of Embedded Software*, Oct. 2001.

[7] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[8] M. H. Klein, T. Ralya, B. Pollak, and R. Obenza, *A Practitioners' Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*. Kluwer Academic Publishers, 1993.

[9] T. Gerdsmeier and R. Cardell-Oliver, "Analysis of Scheduling Behaviour using Generic Timed Automata," *Electronic Notes in Theoretical Computer Science*, vol. 42, 2001.

[10] A. Benveniste, P. Caspi, S. Edwards, N. Halbwachs, P. L. Guernic, and R. de Simone, "The Synchronous Languages 12 Years Later," *Proceedings of the IEEE*, vol. 91, pp. 64–83, 2003.

[11] T. A. Henzinger, B. Horowitz, and C. M. Kirsch, "GIOTTO: A time-triggered language for embedded programming," *Proceedings of the IEEE*, vol. 91, pp. 84–99, 2003.

[12] Y. Abdeddaïm and O. Maler, "Preemptive job-shop scheduling using stopwatch automata," in *Proceedings of TACAS*, 2002, pp. 113–126.

[13] C. Ericsson, A. Wall, and W. Yi, "Timed Automata as Task Models for Event-Driven Systems," in *Proceedings of RTSCA '99*, 1999.

[14] V. Subramonian, C. Gill, C. Sanchez, and H. Sipma, "Reusable Models for Timing and Liveness Analysis of Middleware for Distributed Real-Time Embedded Systems," in *Proceedings of EMSOFT*, October 2006, pp. 252–261.

[15] G. Madl, S. Abdelwahed, and D. C. Schmidt, "Verifying Distributed Real-time Properties of Embedded Systems via Graph Transformations and Model Checking," *Real-Time Systems*, vol. 33, pp. 77–100, Jul 2006.

[16] G. Madl, N. Dutt, and S. Abdelwahed, "Performance Estimation of Distributed Real-time Embedded Systems by Discrete Event Simulations," in *Proceedings of EMSOFT*, October 2007, pp. 183–192.

[17] G. Madl and S. Abdelwahed, "Model-based Analysis of Distributed Real-time Embedded System Composition," in *Proceedings of EMSOFT*, 2005, pp. 371–374.

[18] F. Cassez and K. Larsen, "The impressive power of stopwatches," in *Proceedings of CONCUR*, 2000, pp. 138–152.

[19] P. Pettersson and K. G. Larsen., "UPPAAL2k," *Bulletin of the European Association for Theoretical Computer Science*, vol. 70, pp. 40–44, feb 2000.

[20] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya, "What's decidable about hybrid automata?" *Journal of Computer and System Sciences*, vol. 57, no. 1, pp. 94–124, 1998.

[21] M. Bozga, S. Graf, I. Ober, I. Ober, and J. Sifakis, "The IF Toolset," *Formal Methods for the Design of Real-Time Systems, LNCS 3185*, pp. 237–267, Sep 2004.