Quantitative analysis of real-time systems using priced timed automata

Patricia Bouyer, Uli Fahrenberg, Kim Guldstrand Larsen, Nicolas Markey

Quantitative analysis of real-time systems using priced timed automata

Patricia Bouyer*
bouyer@lsv.ens-cachan.fr

Uli Fahrenberg†
uli@cs.aau.dk

Kim G. Larsen†
kgl@cs.aau.dk

Nicolas Markey*
markey@lsv.ens-cachan.fr

*LSV – CNRS & ENS Cachan
61 avenue du Président Wilson
94230 Cachan – France

†Department of Computer Science
Aalborg Universitet
9220 Aalborg Ø – Denmark

The problems of time-dependent behavior in general, and dynamic resource allocation in particular, pervade many aspects of modern life. Prominent examples range from reliability and efficient use of communication resources in a telecommunication network to the allocation of tracks in a continental railway network, from scheduling the usage of computational resources on a chip for durations of nano-seconds to the weekly, monthly or longer-range reactive planning in a factory or a supply chain.

These problems have been subject to substantial research for decades by different communities such as operational research, computer systems performance evaluation as well as planning and scheduling, witnessed by large ACM communities such as SIGMETRICS and PERFORMANCE. In this paper we argue that the formalism of timed automata together with recent extensions provides an alternative framework with complementary, yet competitive, results in terms of modeling capabilities and efficiency of analysis.

Timing: Twenty years ago, R. Alur and D. Dill introduced the notion of *timed automata*. As a witness for the importance of the formalism one may consider the 2008 Computer-Aided Verification Award given to Alur and Dill for their seminal 1990 article *Automata for modeling real-time systems*,[5] which provided the theoretical foundation for the computer-aided verification of real-time systems.

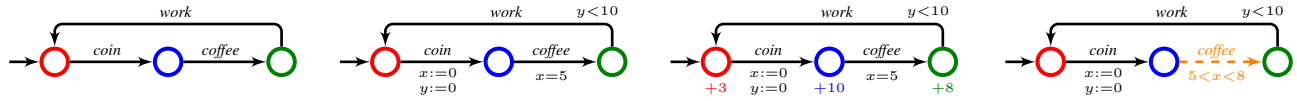
Real-time systems and resource allocation problems have manifested themselves under different names in application domains such as manufacturing, transport, communication networks, embedded systems, and digital circuits, and have been treated using theories and methods in several disciplines. Most of these applications involve distributed, reactive systems of considerable complexity, and with a number of real-time constraints in the sense that correctness not only depends on the logical ordering of events of the systems, but also on the relative timing between these.

State-based models have been the basis of a wide range of successful computer-supported verification methodologies allowing the efficient prediction of functional properties, *e.g.* absence of deadlock or memory overflow. However, many of the models used in this methodology are purely discrete and their treatment of time is purely qualitative, that is, behaviors are just sequences of events appearing one after the other but without any quantitative timing information about the duration of actions and the time between events. Timed automata allow such timing constraints to be expressed, while being amenable to computer-aided analysis methods such as *simulation*, *verification*, *optimization* and *controller synthesis*.

Performance: In all of the above applications, an explicit constraint on timing is only one of a number of quantitative aspects of importance. Within embedded systems additional key quantities include *energy* and *memory* consumption, in communication networks required *band-width* is a key quantity, and within the factory and supply chain applications need for *storage* and overall *cost* for a given production are crucial quantities. The extended notion of *priced* or *weighted* timed automata has been put forward as a formalism allowing for such additional and time-dependent quantities to be modeled, without hampering efficient analysis and even permitting *optimization*.

Uncertainty: Classical models for scheduling in manufacturing, such as job-shop problems, are somewhat detached from industrial practice and reality. They assume that the duration of every step as well as the arrival times are fixed and known with certainty; in practice however, it is rarely the case that a schedule is executed as planned.

For solving problems related to *expected* time and performance properties, *stochastic process models* have been very successful.



(a) A finite automaton (b) A timed automaton (c) A priced timed automaton (d) A timed game

Figure 1: Several refinements of a model (a) of the working mathematician according to Erdős: After insertion of a coin into the coffee dispenser, coffee can be collected, and the scientist can go back to work. In the timed-automaton model (b), precisely five time units pass between coin insertion and coffee collection, and the time which passes between coin insertion and going back to work is less than ten time units. In the priced timed automaton (c), cost rates (modeling *e.g.* energy consumption) are associated with the three states. In the timed game (d), uncertainty as to precisely when coffee is delivered is modeled as an uncontrolled edge.

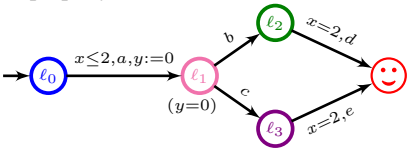
When aiming at *guaranteed* time and performance properties under uncertainty, so-called *timed games* may be used instead. They provide efficient off-line algorithms for synthesizing reactive schedulers with performance guarantees. Such algorithms can plan for the *best* or *worst* case, but the scheduling strategies they produce are adaptive and can take advantage, for example, of the fact that a task has terminated before it was expected to.

In this paper we present the formalism of timed automata and its priced and game extension as a unifying mathematical framework for the modeling, analysis, optimization and synthesis of real-time related phenomena. Figure 1 shows some simple examples of these formalisms; later we provide more elaborate and realistic examples and case studies.

1. TIMED AUTOMATA

1.1 A model for time

Timed automata[5] are a powerful model for representing and reasoning about systems where the notion of time is essential. They are an extension of classical finite-state automata with real-valued variables called *clocks*. These clocks all increase at the same rate, and their values can be used to restrict availability of transitions and how long one can stay in a location (or state). Also, clocks can be reset to zero when a transition is taken. To this end, each transition has associated with it a *guard* (which must be satisfied for the transition to be enabled) and a set of clocks to be reset, and each location carries an *invariant* which must be continuously satisfied when the system is in the location. Below we show an example of a timed automaton with two clocks x and y, and label set $\{a, b, c, d, e\}$. Note that no time can elapse in location ℓ_1 due to the invariant $(y = 0)$; locations with this property are called *urgent*.



Guards and invariants are given as comparisons $x \leq \alpha$ or $x < \alpha$, or the reverse relations, of a clock value with an integer constant, or as conjunctions of these. Sometimes also so-called *diagonal* constraints $x - y \leq \alpha$ (or $<$ or other) are allowed, but other extensions quickly lead to undecidability issues, see below.

A configuration of the system is made of a location and a clock valuation (in our case, values for both clocks x and y). A possible execution in our example is:

$$\begin{bmatrix}\ell_0\\0\\0\end{bmatrix} \xrightarrow[1.3]{\text{delay}} \begin{bmatrix}\ell_0\\1.3\\1.3\end{bmatrix} \xrightarrow[a]{\text{action}} \begin{bmatrix}\ell_1\\1.3\\0\end{bmatrix} \xrightarrow[c]{\text{action}} \begin{bmatrix}\ell_3\\1.3\\0\end{bmatrix} \xrightarrow[0.7]{\text{delay}} \begin{bmatrix}\ell_3\\2\\0.7\end{bmatrix} \xrightarrow[e]{\text{action}} \begin{bmatrix}\text{☺}\\2\\0.7\end{bmatrix}$$

where the first component of a configuration is the location and the second and third components give the values of clocks x and y, respectively. This execution corresponds to a delay of 1.3 time units in ℓ_0, the firing of transition a (which is enabled because the value of clock x is less than 2; clock y is then set to 0), the firing of transition c (which occurs without delay as ℓ_1 is urgent), *etc.*

In the context of verification, several problems are of interest, like the model-checking of safety properties ("*Can a distinguished set of states be avoided?*"), reachability/liveness properties ("*Can/will a distinguished goal state be reached?*"), or more involved properties such as response properties ("*Is any request eventually granted?*"). As a model for real-time systems, these properties can include quantitative constraints, for instance time-bounded reachability, or time-bounded response properties ("*Is any request granted within two minutes?*"). It is also relevant to compute optimal time bounds for these properties, *e.g.* optimal-time reachability ("*What is the minimum time required for reaching a distinguished set of states?*").

1.2 The region abstraction

A timed automaton is a syntactical representation of an infinite transition system, since clocks take (nonnegative) real values. However, there is a way to deal with this infinity of configurations by reasoning symbolically: the main theoretical ingredient for solving problems on timed automata is the notion of *regions*,[5] which provide a finite partitioning of the state space such that states within a given region are *bisimilar*, *i.e.* behaviorally indistinguishable.

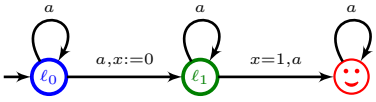
The precise definition of regions is such that inside a region, integral parts of clock values do not change, and also the ordering of clocks according to their values' fractional parts stays the same. Special consideration has to be given to the cases where one or more clock values are integers, and finiteness of the region partitioning is ensured by considering as equivalent all clock values which exceed the maximal constant appearing in guards and invariants of the timed automaton in question. In the left part of Figure 2 we show the forty-four regions for two clocks x and y with maximal constant equal to 2. In this two-clock case, regions can be points (both clocks have integer values), open line segments (one clock has integer value, or their fractional parts are equal), open triangles, or open unbounded rectangles.

From two equivalent configurations (same location, region equivalent valuations), by delaying or by taking a transition, similar regions will be visited and similar behaviors will be possible. Regions are thus a way to finitely abstract the behaviors of a timed automaton. There are finitely many regions, and by considering as abstract configurations pairs of locations and regions, we get a finite automaton, called the *region automaton*, which preserves many properties including reachability, liveness and safety. Hence verification of those properties on the original timed automaton can be transferred to the finite region automaton and then checked using standard algorithms.

1.3 The limits of the region abstraction

Not all properties can be decided on timed automata using the region abstraction, and problems such as checking inclusion ("*Are all real-time behaviors of a timed automaton also behaviors of another timed automaton?*") and universality ("*Can all real-time behaviors be realized in a given timed automaton?*") are undecidable.

Also, the set of real-time behaviors exhibited by timed automata is not closed under complement, and not all timed automata are determinizable. As a counterexample for these properties, one can use the following timed automaton:



It accepts all behaviors with at least two a's separated by one time unit. It can be shown that no deterministic timed automaton exists with exactly the same behaviors, and also that no timed automaton can implement precisely all complementary behaviors.

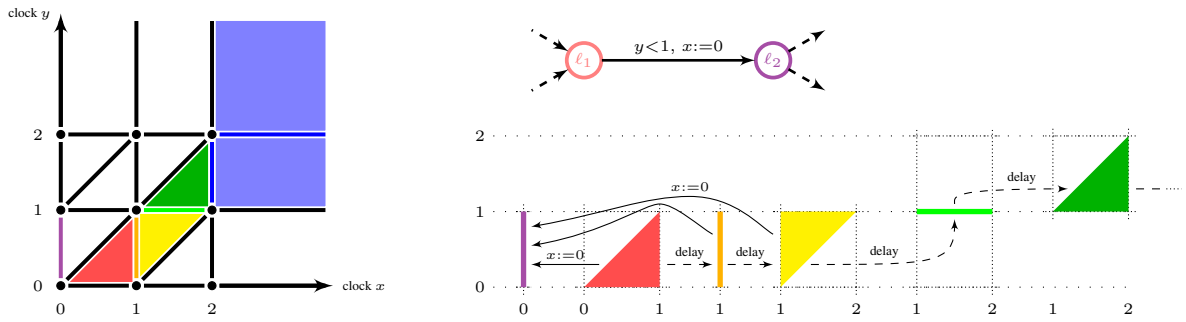


Figure 2: The region abstraction is a finite representation of all possible behaviors of the timed automaton. Consider the timed automaton on top of the picture, and assume we enter location ℓ_1 with clock values (x_0, y_0) for which $0 < y_0 < x_0 < 1$ (a point in the red triangle, see the picture on the left); as clock y has value strictly less than 1, we have the option to switch to location ℓ_2, which would reset clock x and end up in the purple region. We also have the option to delay in ℓ_1; in that case, we exit the red triangle and reach the orange line. Here again we have two options: switching to ℓ_2, or delaying to the yellow clock region. In case we still decide to wait in that region, we reach the green line. From that region on, the transition to ℓ_2 is not enabled anymore. This description of the possible behaviors starting from the red region, which has been represented on the picture to the right, does not depend on the precise values of the clocks: region equivalence preserves enough information to encode exactly the behaviors of the underlying timed automaton.

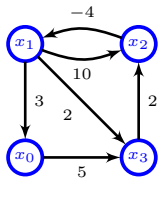
1.4 Timed automata in practice

The region abstraction is a powerful tool for showing decidability of a number of interesting properties, but unfortunately, the region-based verification algorithms introduced above are infeasible in practice. The number of regions grows exponentially with the number of clocks, hence these algorithms have exponential time complexity. Using *on-the-fly* techniques, one can reduce the complexity to polynomial-space, and indeed it can be shown[4] that *e.g.* the reachability problem is PSPACE-complete.

Algorithms which have shown to be feasible, even efficient, in practice are based on the so-called *zone graph* abstraction:[30] A *zone* is a set of clock valuations defined by a clock constraint and can hence be represented by such; the zone graph has as vertices pairs of locations and zones which satisfy the location's invariant, and its edges are derived from the transitions of the given timed automaton. The number of zones is unbounded, so unlike the region graph, the zone graph is infinite. Finiteness can be enforced using a technique known as normalization;[12] however the number of zones is still much larger than the number of regions, and moreover the same zone can be represented using many different clock constraints.

The reason for zone-based algorithms to be efficient in practice is two-fold: First, the algorithms used have no need to explore all of the zone graph (they work *on-the-fly*), and zones are commonly bigger than regions, hence the part of the zone graph to be explored is smaller. Second, operations on zones can be implemented very efficiently (in time cubic in the number of clocks): Zones are usually represented using *difference-bound matrices*, or DBMs. The DBM representation of a zone on a set of k clocks has $(k+1)\times(k+1)$ entries, where an entry $c_{i,j}$ represents a clock constraint $x_i - x_j \leq c_{i,j}$ and an extra clock x_0 is added to represent absolute clock constraints $x_i \leq c_{i0}$. DBMs in turn can be represented as directed weighted graphs; see below for an example of a zone and its DBM (graph) representation. Canonical representations of zones can be obtained using shortest-path closure or shortest-path reduction of their DBM graphs, and delay and reset operations on zones can be efficiently implemented on the DBM representations.

$$Z = \begin{cases} x_1 \leq 3 \\ x_1 - x_2 \leq 10 \\ x_1 - x_2 \geq 4 \\ x_1 - x_3 \leq 2 \\ x_3 - x_2 \leq 2 \\ x_3 \geq -5 \end{cases}$$

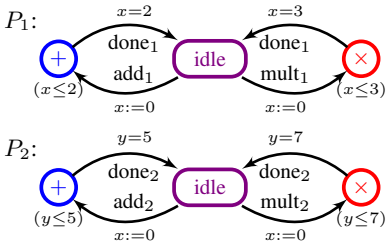


1.5 Task graph scheduling: time optimality

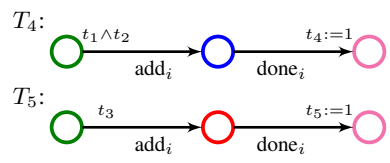
A task graph problem involves a number of tasks $T_1, ..., T_m$, a number of machines or processors $P_1, ..., P_n$, and a (partial) mapping d giving, for each task T_i and processor P_j, the time $d(i,j)$ for computing T_i on P_j. In addition there is a partial order on the tasks used for describing dependencies. Figure 3 is an example of a task graph problem.

We want to determine a schedule of when to start the execution of tasks, and on which processors, that minimizes the total execution time while being feasible in respecting the following conditions: (a) a task can be executed only if all its predecessors have completed; (b) each machine can process at most one task at a time; (c) tasks cannot be pre-empted.

Task graph scheduling problems may be easily modeled as networks of timed automata so that every run corresponds to a feasible schedule and the fastest run gives the time-optimal schedule: For each processor we construct a small timed automaton able — when idle— to handle within the appropriate amount of time the requests from the tasks. For the processors of Figure 3, these are as follows:



Each task is modeled as a timed automaton waiting to be served by either of the processors, conditioned by the completion of its predecessors (indicated by Boolean variables t_1 through t_5). Tasks T_4 and T_5 of our example can be represented as follows:



Extensive experiments on benchmarks have demonstrated that the above timed automata approach to task graph scheduling is competitive compared with more

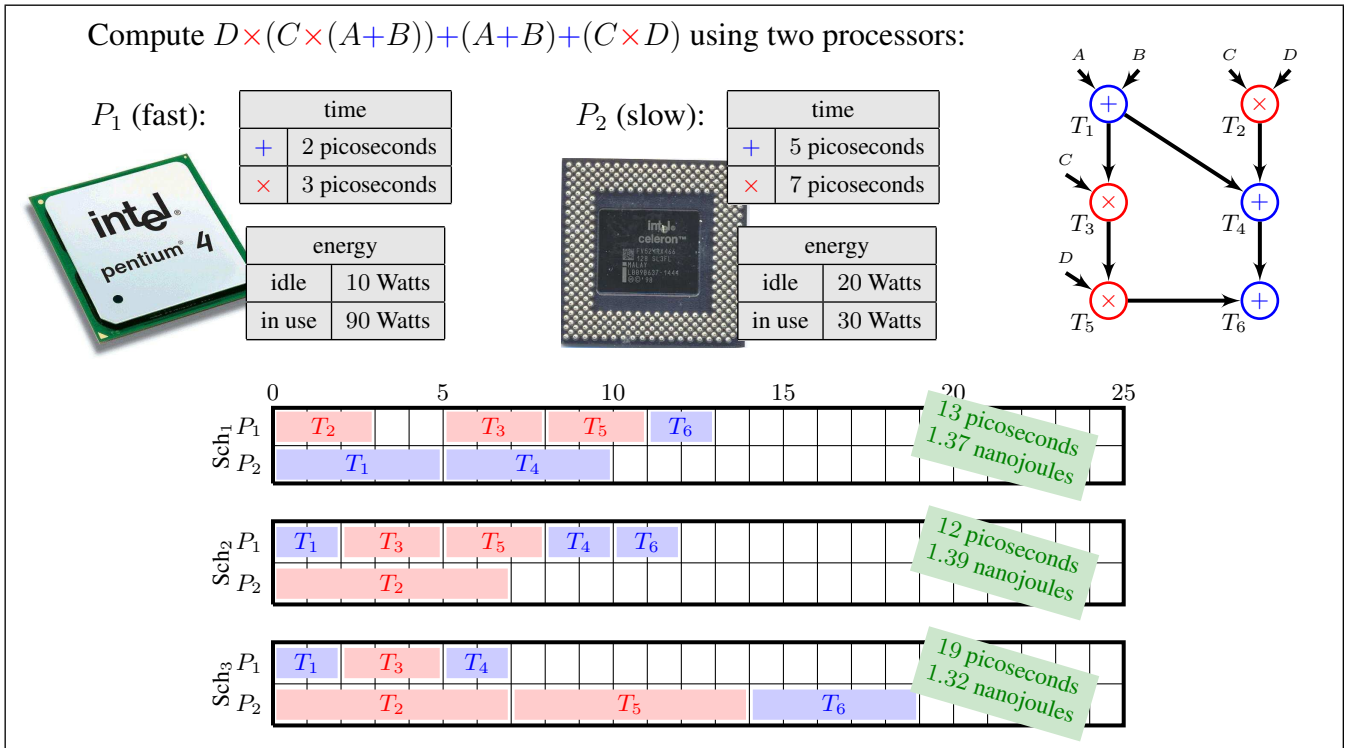


Figure 3: Task graph problem with 6 tasks, where each task corresponds to the computation of a given sub-expression of the term $(D \times (C \times (A + B)) + ((A + B) + (C \times D))$. Given the execution platform with two processors, P_1 and P_2, and corresponding computation times for addition and multiplication, as well as their energy consumption, Sch_1 to Sch_3 provide three feasible schedules, where Sch_2 is in fact time-optimal, and Sch_3 is energy-optimal.

traditional approaches from operations research (*e.g.* mixed-integer linear programming) as well as specialized, heuristic algorithms from planning and scheduling.[1] Furthermore the generic approach of timed automata admits easy incorporation of more specialized features (*e.g.* release times, deadlines) to the models and scheduling.

1.6 Extensions of timed automata

As we shall see in Section 4, timed automata are a rich extension of classical automata with efficient tool support and several successful industrial applications. As such, they are often cited as the model of choice for representing and reasoning about embedded and real-time systems.

This success has led to several extensions of the model, for instance with more general guards or resets being allowed (*e.g.* additive guards[11] or non-deterministic updates of clocks[12]), or with more involved dynamics measuring other quantities than time. Unfortunately, these extensions quickly lead to undecidability; *e.g.* for timed automata in which clocks can be stopped (so-called *stopwatch automata*), even basic properties such as safety or liveness are undecidable.[29]

On the other hand, the model of *hybrid automata*[29] though suffering from the same undecidability problems as mentioned for other classes above, has emerged as a popular formalism for which semi-decision and approximation procedures have been developed. The model of *priced timed automata*, which we shall discuss in the next section, form an intermediate class between timed and hybrid automata for which some of the good decidability properties of timed automata are retained. Other intermediate classes of models have been investigated, including *linear hybrid automata*[29] and *integration graphs*,[33] providing semi-decidability in general and decidability under certain restrictions.

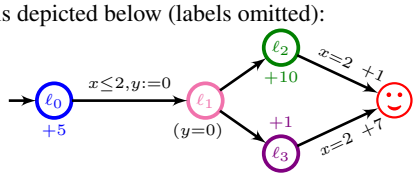
2. PRICED TIMED AUTOMATA

2.1 A model for resources

Time is not the only quantitative notion of interest when designing embedded systems; other quantities such as energy or memory consumption, required bandwidth, or accumulated cost can be important to measure in such systems.

These notions are intimately connected to time, because the longer the device is operating, the more resources it consumes. This makes timed automata the model of choice to reason about those quantities, and has led to the definition of *priced timed automata*,[6,10] extending timed automata with *cost* (which is the general name we will use in the sequel to refer to the various quantities which can be modeled within this formalism; in some other literature, this is referred to as *reward*).

A priced timed automaton is hence a timed automaton with extra information indicating how the cost is evolving in locations and during transitions. To avoid the undecidability problems of hybrid automata, cost information cannot be used to guard transitions; the cost is only an *observer variable*, and whether a transition is enabled only depends on timing information, not cost value. An example of a priced timed automaton, extending the timed automaton of the previous section, is depicted below (labels omitted):



A decoration $+10$ on a location indicates that cost increases by 10 units per time unit in the location; a decoration $+7$ on a transition indicates that taking the transition increases

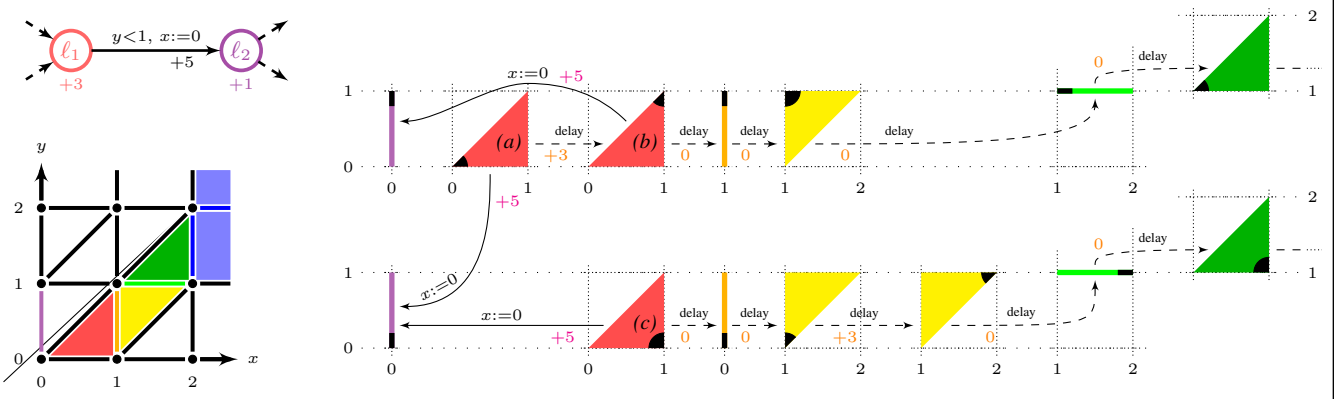


Figure 4: The corner-point abstraction refines the region abstraction by also keeping track of the corner point close to which an execution runs. This is needed to measure costs: for instance, if we are in location ℓ_1 and in the red region where $0 < y < x < 1$, the price of delaying depends on the value of the clocks. From *(a)*, where both x and y are arbitrarily close to 0, we can let almost one time-unit elapse and reach *(b)*. The resulting cost is arbitrarily close to $+3$. On the other hand, from *(c)*, where x and y are arbitrarily close to 1 and 0, respectively, letting time elapse takes us to the subsequent region, so that the cost is arbitrarily close to 0. (Notice that for readability, some resetting transitions have been omitted.)

overall cost by 7 units (locations and transitions without cost indication have cost 0). The executions of such an automaton are those of the underlying timed automaton. The total cost of the example execution given in Section 1.1 (delaying 1.3 time units in ℓ_0, 0.7 time units in ℓ_3, and ending in the rightmost location) can be computed as

$$1.3 \times (+5) + 0.7 \times (+1) + 7 = 14.2$$

2.2 Optimizing the resources

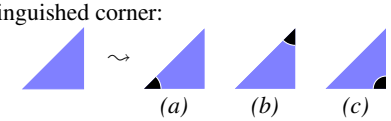
Natural optimization questions can be posed on that model, *e.g.* the *optimal reachability* problem (minimum cost for reaching a given goal), the *mean-cost optimization* problem (mean cost used in the long run), or the *discounted-cost optimization* problem (where costs are discounted exponentially as time elapses).

As an example, we compute the minimum cost that is required for reaching location ☺ in the previous example. There are two families of executions: those that go through ℓ_2 and those that go through ℓ_3. Furthermore, in each family, there is a single parameter t: the time elapsed in location ℓ_0; everything else is determined by the guards in the automaton. Hence the minimum cost is:

$$\inf_{0 \le t \le 2} \min \begin{pmatrix} 5t+10(2-t)+1 \\ 5t+(2-t)+7 \end{pmatrix} = 9 \quad (1)$$

where the expressions $5t + 10(2 - t) + 1$ and $5t + (2 - t) + 7$ give the cost of executions going through ℓ_2 respectively ℓ_3 after delaying t time units in location ℓ_0.

The standard region construction is not accurate enough to properly keep track of cost information, and a refinement of the region abstraction, the *corner-point abstraction*,[13] has to be used to solve the optimization problems mentioned above. For this abstraction, regions are refined by distinguishing their corner points. As an example, the two-dimensional region depicted below is refined into three region-corner pairs; the meaning of a region-corner pair is that the current clock valuation is arbitrarily close to the distinguished corner:



Similar to the refinement of regions, the transitions in the region automaton have to be refined to keep track of the corners. In the example above, there is a (delay) transition from region-corner pair *(a)* to *(b)*, whereas *(c)* cannot be reached neither from *(a)* nor from *(b)*. Figure 4 illustrates the corner-point abstraction of an example priced timed automaton. This graph has two types of delay edges: either within a region, from one corner to another one, or from a corner of a region to the corresponding corner in the subsequent region. The first case corresponds to a delay of "almost" one time unit, while the second case corresponds to a delay of "almost" zero time units. In addition, there are edges representing transitions of the timed automaton (which reset clock x in our example of Figure 4). In that case as well, there is a natural mapping between corners.

The edges of the corner-point abstraction are labeled with discrete cost information: if the cost rate in the current location is $+3$, all one time-unit edges have label $+3$, and all zero time-unit edges get label 0. Edges coming from discrete transitions are labeled with the cost of the transition ($+5$ in the example).

The corner-point abstraction can be used to solve many optimization problems, as it can be shown that in these cases, optimal total cost is obtained for runs which always take transitions close to integer clock values. Hence the optimization problem reduces to a problem on a finite graph which can be solved using different standard techniques. This is the case for the mean-cost optimization problem[13] and the discounted-cost problem.[25] For optimal reachability, another technique (*priced regions*) has been used[10] which also extends to a setting of more than one cost variable.[35]

As for algorithm and tool support, the zone-based approach has been successfully extended to solve the optimal reachability problem,[34] by introducing *priced zones*, and tool support is available in UPPAAL CORA. For mean-cost and discounted-cost optimization, active research is being conducted in developing efficient zone-based algorithms, or alternatively showing that no such algorithms exist.

2.3 Task graph scheduling: energy optimality

Reconsidering our running task graph scheduling problem of Section 1.5, cost-optimal reachability for priced timed automata may be used to provide energy-optimal schedules. The energy needed for performing computation steps is modeled as cost in the priced timed automaton model, and optimal reachability techniques can be used for finding an energy-optimal schedule.

For the task graph scheduling instance of Figure 3, energy consumption of the two processors is reflected in the respective timed automata by suitable cost-rates in the locations corresponding to the processor being idle or in use. The processors can then be represented by the following two priced timed automata:

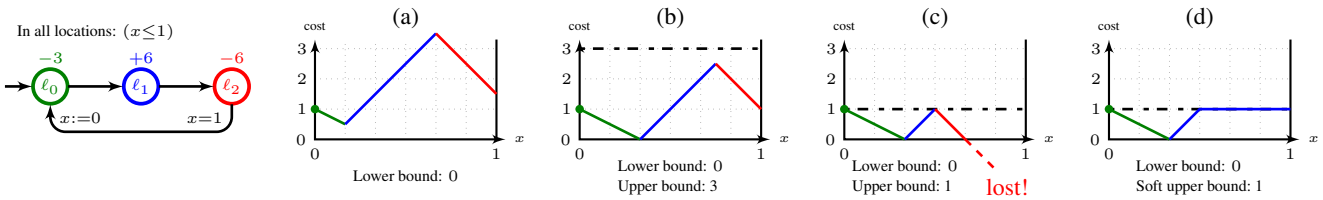
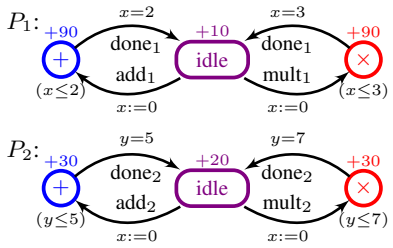


Figure 5: The resource management problem asks whether it is possible to maintain the cost level within fixed bounds. There can be a lower bound only (a), a lower and an upper bound (b, c), or a lower bound and a soft upper bound above which cost level cannot increase. Figures (a), (b) and (d) represent solutions to the respective problems for the priced timed automaton depicted on the left: there is an infinite run that satisfies the global constraint. In case (a) for instance, we have depicted a possible schedule for the first cycle, and this run can be repeated because at the start of the second cycle, the cost level is larger than at the start of the first cycle. In figure (c), the proposed schedule violates the lower bound, and it can be shown that there exists no infinite run which maintains cost level within the specified bounds.



2.4 Managing the resources

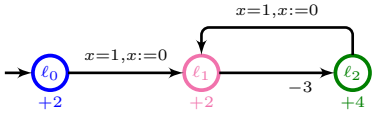
Up to this point we have only employed priced timed automata as a formalism for modeling time-dependent *consumption* of resources. However, in several situations resources may not just be consumed but also occasionally regained, *e.g.* in autonomous robots with rechargeable batteries, or in tanks which may not only be emptied but also filled. Extending priced timed automata to allow for both positive (regaining) and negative (consumption) rates provides a natural modeling formalism.[21]

However, a new question now emerges related to the appropriate management of resources: "*Is it possible to maintain the level of resources within fixed bounds?*" Such resource-bound problems are highly relevant to the analysis of several embedded systems, *e.g.* it is natural to plan the usage of a device with rechargeable batteries so that one never runs out of energy, nor exceeds the maximum capacity for energy storage. Figure 5 shows a priced timed automaton together with some resource management problems.

Few results have been obtained on this problem so far: only the case of *one-clock* priced timed automata has been investigated.[14] This restriction has two important consequences: Cycle detection can be done statically, as each resetting transition leads to a configuration with clock value 0, and the region automaton can be coarsened so that the partition consists of intervals with endpoints given by the constants in the automaton's guards. As a consequence, there are only polynomially many regions.

Under the additional assumption that the cost cannot be updated during transitions (hence cost evolves only in locations), it can be shown[14] that for finding runs which satisfy a global lower-bound constraint, with or without soft upper bound, one can restrict oneself to look for runs with *integral* delays. Hence the corner-point abstraction can be used for this, and the problems are solvable in polynomial time.

For priced timed automata with more than one clock, no results are known, but even for one-clock automata with cost updates during transitions, there are some difficulties which mean that abstractions like the above are insufficient. As an example, consider the following priced timed automaton:

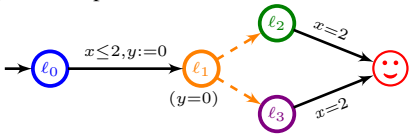


Assuming that we start with initial cost 0, this automaton has exactly one feasible execution in which the cost level remains non-negative: after spending 1 time unit in location ℓ_0, we alternately spend half a time unit in ℓ_1 and half a time unit in ℓ_2. Any other execution eventually violates the lower bound. Hence in this case, runs satisfying the lower bound cannot be found using the corner-point abstraction.

3. PRICED TIMED GAMES

3.1 A model for uncertainties

The systems we have considered so far are *closed* in the sense that we have a complete description of the system. This is not sufficient to model embedded systems where interaction with the environment is crucial, or systems with some imprecision. These can be modeled using (two-player) *timed games*,[8] in which some actions are triggered by the environment (we can think of signals received by sensors, or of unexpected events). The aim is to *control*, or *guide*, the system so that it will be safe or correct regardless of the way the environment interferes. An example of a timed game is depicted below:



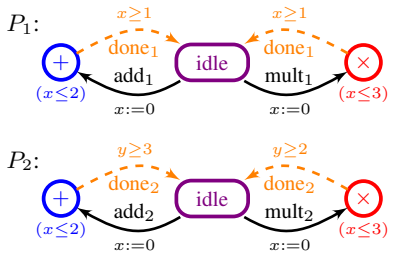
Dashed edges belong to the environment (they are *uncontrollable*): when they are fireable, the system cannot prevent (nor force) them to be fired. Here, the system cannot decide whether it goes through ℓ_2 or through ℓ_3.

For simple correctness criteria, *e.g.* reachability or safety, the set of *winning states* (*i.e.*, states from which the system can be controlled under the safety constraint) and also *winning strategies* (*i.e.*, policies for how to control the system) can be computed using the region abstraction.[8] Also computability of time-optimal strategies,[7] as well as strategies under partial observability, has been demonstrated. For the latter, decisions are based on discrete observations giving only partial information of the system state, depending on the availability and precision of sensors.[19] For efficient algorithms, a zone-based approach for solving timed games with reachability and safety objectives has been developed,[18,38] and tool support is available in UPPAAL-TIGA.

3.2 Task graph scheduling: timing uncertainty

Returning to our running task graph scheduling example, we can use the formalism of timed games to model uncertainty in precisely how much time a certain computation on a given processor takes. In Section 1.5 we modeled computation times by precise numbers, whereas we now can make the model more realistic by only providing *interval bounds* within which computation times are prescribed to lie. The timed game models below provide version of the processors P_1 and P_2 from Figure 3 in which computation times are prescribed to lie in the intervals $[1, 2]$ for addition and $[1, 3]$ for multi-

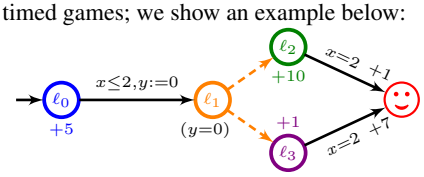
plication on P_1, and similarly for P_2.



Using these models, a computed time-optimal schedule will no longer be a simple fixed assignment of tasks and time slots to processors, but rather a flexible dynamic assignment, where task scheduling can be adapted on-line according to actual completion times of previous tasks. (Hence we cannot display the solution here.)

3.3 Cost-optimal strategies

It is natural to extend the timed game framework with cost information, hence making it possible to model uncertainty as well as resource use, and to ask for *controllability under resource constraints*, or for *optimal* controllability. The model of *priced timed games* is a synthesis of priced timed automata and timed games; we show an example below:



In this example we may *e.g.* want to compute the minimum cost for reaching location ☺ regardless of the moves of the environment (which is in charge of the edges out of ℓ_1 as before). As the system cannot control whether execution goes through ℓ_2 or ℓ_3, the minimum cost is given by the term

$$\inf_{0 \leq t \leq 2} \max \begin{pmatrix} 5t+10(2-t)+1 \\ 5t+(2-t)+7 \end{pmatrix} = 14 + \frac{1}{3}$$

where t is the delay spent in location ℓ_0. Solving this, one arrives at a minimum cost of $14\frac{1}{3}$ which is attained for $t = \frac{4}{3}$. As this is not an integer, one sees that techniques based on the corner-point abstraction are not sufficient for computing optimal-reachability strategies, even in case of one-clock priced timed games.

Generally, priced timed games are much more difficult to analyze than priced timed automata. Using reductions from the Halting problem for two-counter machines, one can show that cost-optimal strategies are undecidable,[17] even when restricted to priced timed games with only three clocks.

Decidability has been shown for classes of priced timed games with strong conditions on the cost evolution[3] and for one-clock priced timed games.[15] The reason for the latter is the same as for one-clock priced timed automata above: resetting the clock leads to a configuration with a known clock valuation.

4. APPLICATIONS & TOOLS

Timed automata and their extensions have been applied to the modeling, analysis and optimization of numerous real-time applications. In this section we give a few examples, not aiming at being exhaustive but rather to illustrate the wide range of application domains.

A variety of mature tools are available which provide important computer-aided support for applications. Well-known tools include UPPAAL, KRONOS, and HYTECH, but there is a large number of other tools available. The electronic version of this article contains an extra section which aims to give an overview together with references to the individual tools.

The timed automata formalism is now routinely applied to the modeling and analysis of real-time control programs, including a wide class of Programmable Logic Controller (PLC) control programs[23,36] and timing analysis and code generation of vehicle control software,[39] and the timed automaton approach has also demonstrated its viability to the timing analysis of certain classes of asynchronous circuits.[16]

Similarly, numerous real-time communication protocols have been analyzed using timed automata technology, often with inconsistencies being revealed: *e.g.* using real-time model checking, the cause of a 10 year old bug in the IR-link protocol used by Bang&Olufsen was identified and corrected.[27] Most recently, real-time model checking has been applied to the clock synchronization algorithm currently used in a wireless sensor network that has been developed by the Dutch company CHESS.[37] Here it is shown that in certain cases a static, fully synchronized network may eventually become unsynchronized if the current algorithm is used, even in a setting with infinitesimal clock drifts.

During the last years, timed automata modeling of multitasking applications running under real-time operating systems has received substantial research effort. Here the goals are multiple: to obtain less pessimistic worst-case response time analysis compared with classical methods for single-processor systems;[40] to relax the constraints of period task arrival times of classical scheduling theory to task arrival patterns that can be described using timed automata;[26] to allow for schedulability analysis of tasks in terms of concurrent objects executing on multiprocessor or distributed platforms (*e.g.* MPSoC).[22]

Just as symbolic reachability checking of finite-state models has lead to very efficient planning and scheduling algorithms, reachability checking for (priced) timed automata has demonstrated competitive and complementary performance with respect to classical approaches such as MIPL on optimal scheduling problems involving real-time constraints, *e.g.* job-shop and task-graph scheduling[1,9] and aircraft landing problems.[34] In fact a translation of the variant PDDL3 of PDDL (Planning Domain Definition Language) into priced timed automata has been made[24] allowing optimal planning questions to be answered by cost-optimal reachability checking. Industrial applications include planning a wafer scanner from semiconductor industry[28] and computation of optimal paper paths for printers.[31]

Most recently, computation of winning strategies for timed games has been applied to controller synthesis for embedded systems, including synthesis of most general non-preemptive online schedulers for real-time systems with sporadic tasks,[2] synthesis of climate control for pig stables provided by the company Skov A/S,[32] and automatic synthesis of robust and near-optimal controllers for industrial hydraulic pumps.[20]

5. CONCLUSIONS

Timed automata and their priced and game extensions provide a uniform and expressive formalism for dynamic resource allocation problems with hard real-time constraints, *i.e.* timing constraints that must be satisfied under all circumstances. This is in contrast to soft real-time constraints, which only need to be met with a certain probability, .999 say, and which require stochastic modeling formalisms such as discrete-time or continuous-time Markov chains, queueing models, *etc*. While hard real-time focuses on worst-case analysis, soft real-time addresses more refined properties such as average-case performance.

However within the setting of hard real-time, timed automata and their extensions allow for analysis of a wide collection of performance and optimization problems, with results competitive with respect to more traditional approaches such as mixed-integer linear programming or others.

Particularly challenging problems remaining to be settled include decidability of synthesis for priced timed games under partial observability, as well as a range of resource management problems in the setting of priced timed automata and games with both consumption and regaining of resources.

Acknowledgments

The authors are partly supported by the European project QUASIMODO (FP7-ICT-STREP-214755). The French authors are supported by project DOTS (ANR-06-SETI-003). The Danish authors are supported by the Danish Center of Excellence MT-LAB.

6. REFERENCES

[1] Y. Abdeddaïm, E. Asarin, and O. Maler. Scheduling with timed automata. *Theoretical Computer Science*, 354(2):272–300, 2006.

[2] K. Altisen et al. A framework for scheduler synthesis. In *IEEE Real-Time Systems Symposium*, p. 154–163, 1999.

[3] R. Alur, M. Bernadsky, and P. Madhusudan. Optimal reachability in weighted timed games. In *Proc. 31st International Colloquium on Automata, Languages and Programming (ICALP'04)*, Lecture Notes in Computer Science 3142, p. 122–133. Springer, 2004.

[4] R. Alur, C. Courcoubetis, and D. L. Dill. Model-checking for real-time systems. In *Proc. 5th Annual Symposium on Logic in Computer Science (LICS'90)*, p. 414–425. IEEE Computer Society Press, 1990.

[5] R. Alur and D. L. Dill. Automata for modeling real-time systems. In *Proc. 17th International Colloquium on Automata, Languages and Programming (ICALP'90)*, Lecture Notes in Computer Science 443, p. 322–335. Springer, 1990.

[6] R. Alur, S. La Torre, and G. J. Pappas. Optimal paths in weighted timed automata. In *Proc. 4th International Workshop on Hybrid Systems: Computation and Control (HSCC'01)*, Lecture Notes in Computer Science 2034, p. 49–62. Springer, 2001.

[7] E. Asarin and O. Maler. As soon as possible: Time optimal control for timed automata. In *Proc. 2nd International Workshop on Hybrid Systems: Computation and Control (HSCC'99)*, Lecture Notes in Computer Science 1569, p. 19–30. Springer, 1999.

[8] E. Asarin et al. Controller synthesis for timed automata. In *Proc. IFAC Symposium on System Structure and Control*, p. 469–474. Elsevier Science, 1998.

[9] G. Behrmann et al. Efficient guiding towards cost-optimality in UPPAAL. In *Proc. 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'01)*, Lecture Notes in Computer Science 2031, p. 174–188. Springer, 2001.

[10] G. Behrmann et al. Minimum-cost reachability for priced timed automata. In *Proc. 4th International Workshop on Hybrid Systems: Computation and Control (HSCC'01)*, Lecture Notes in Computer Science 2034, p. 147–161. Springer, 2001.

[11] B. Bérard and C. Dufourd. Timed automata and additive clock constraints. *Information Processing Letters*, 75(1–2):1–7, 2000.

[12] P. Bouyer. Forward analysis of updatable timed automata. *Formal Methods in System Design*, 24(3):281–320, 2004.

[13] P. Bouyer, E. Brinksma, and K. G. Larsen. Optimal infinite scheduling for multi-priced timed automata. *Formal Methods in System Design*, 32(1):2–23, 2008.

[14] P. Bouyer et al. Infinite runs in weighted timed automata with energy constraints. In *Proc. 6th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'08)*, Lecture Notes in Computer Science, Lecture Notes in Computer Science. Springer, 2008.

[15] P. Bouyer et al. Almost optimal strategies in one-clock priced timed automata. In *Proc. 26th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'06)*, Lecture Notes in Computer Science 4337, p. 345–356. Springer, 2006.

[16] M. Bozga et al. Verification of asynchronous circuits using timed automata. *Electronic Notes in Theoretical Computer Science*, 65(6), 2002.

[17] Th. Brihaye, V. Bruyère, and J.-F. Raskin. On optimal timed strategies. In *Proc. 3rd International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'05)*, Lecture Notes in Computer Science 3821, p. 49–64. Springer, 2005.

[18] F. Cassez et al. Efficient on-the-fly algorithms for the analysis of timed games. In *Proc. 16th International Conference on Concurrency Theory (CONCUR'05)*, Lecture Notes in Computer Science 3653, p. 66–80. Springer, 2005.

[19] F. Cassez et al. Timed control with observation based and stuttering invariant strategies. In *Proc. 5th International Symposium on Automated Technology for Verification and Analysis (ATVA'07)*, Lecture Notes in Computer Science 4762, p. 192–206. Springer, 2007.

[20] F. Cassez et al. Automatic synthesis of robust and optimal controllers – an industrial case study. In *Proc. 12th International Workshop on Hybrid Systems: Computation and Control (HSCC'09)*, Lecture Notes in Computer Science 5469. Springer, 2009.

[21] A. Chakrabarti et al. Resource interfaces. In *Proc. 3rd International Workshop on Embedded Software (EMSOFT'03)*, Lecture Notes in Computer Science 2855. Springer, January 2003.

[22] A. David et al. Model-based framework for schedulability analysis using UPPAAL 4.1. In G. Nicolescu and P. J. Mosterman, editors, *Model-Based Design for Embedded Systems*, chapter 4, p. 93–119. CRC Press, 2009.

[23] H. Dierks. PLC-automata: a new class of implementable real-time automata. *Theoretical Computer Science*, 253(1):61–93, 2001.

[24] H. Dierks. Finding optimal plans for domains with continuous effects with UPPAAL CORA. In *Proc. ICAPS'05 Workshop on Verification and Validation of Model-Based Planning and Scheduling Systems*, 2005.

[25] U. Fahrenberg and K. G. Larsen. Discount-optimal infinite runs in priced timed automata. *Electronic Notes in Theoretical Computer Science*, 239:179–191, 2009.

[26] E. Fersman et al. Schedulability analysis of fixed-priority systems using timed automata. *Theoretical Computer Science*, 354(2):301–317, 2006.

[27] K. Havelund et al. Formal modeling and analysis of an audio/video protocol: An industrial case study using UPPAAL. In *Proc. 18th IEEE Real-Time Systems Symposium (RTSS'97)*, p. 2–13. IEEE Computer Society Press, 1997.

[28] M. Hendriks, B. van den Nieuwelaar, and F. W. Vaandrager. Model checker aided design of a controller for a wafer scanner. *STTT*, 8(6):633–647, 2006.

[29] Th. A. Henzinger et al. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57(1):94–124, 1998.

[30] Th. A. Henzinger et al. Symbolic model-checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.

[31] G. Igna et al. Formal modeling and scheduling of datapaths of digital document printers. In *Proc. 6th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'08)*, Lecture Notes in Computer Science 5215, p. 170–187. Springer, 2008.

[32] J. J. Jessen et al. Guided controller synthesis for climate controller using UPPAAL-TIGA. In *Proc. 5th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'07)*, Lecture Notes in Computer Science 4763, p. 227–240. Springer, 2007.

[33] Y. Kesten et al. Decidable integration graphs. *Information and Computation*, 150(2):209–243, 1999.

[34] K. G. Larsen et al. As cheap as possible: Efficient cost-optimal reachability for priced timed automata. In *Proc. 13th International Conference on Computer Aided Verification (CAV'01)*, Lecture Notes in Computer Science 2102, p. 493–505. Springer, 2001.

[35] K. G. Larsen and J. I. Rasmussen. Optimal reachability for multi-priced timed automata. *Theoretical Computer Science*, 390(2-3):197–213, 2008.

[36] A. Mader and H. Wupper. Timed automaton models for simple programmable logic controllers. In *Proc. 11th Euromicro Conference on Real-Time Systems (ECRTS'99)*, p. 106–113. IEEE Computer Society, 1999.

[37] M. Schuts et al. Modelling clock synchronization in the Chess gMAC WSN protocol. *CoRR*, abs/0912.1901, 2009.

[38] S. Tripakis and K. Altisen. Controller synthesis for discrete and dense-time systems. In *Proc. World Congress on Formal Methods in the Development of Computing Systems (FM'99)*, Lecture Notes in Computer Science 1708, p. 233–252. Springer, 1999.

[39] S. Tripakis and S. Yovine. Timing analysis and code generation of vehicle control software using TAXYS. *Electronic Notes in Theoretical Computer Science*, 55(2), 2001.

[40] L. Waszniowski and Z. Hanzálek. Formal verification of multitasking applications based on timed automata model. *Real-Time Systems*, 38(1):39–65, 2008.