REMES: A Resource Model for Embedded Systems

Cristina Seceleanu Aneta Vulgarakis Paul Pettersson
MRTC, Mälardalen University, Västerås, Sweden
{cristina.seceleanu,aneta.vulgarakis,paul.pettersson}@mdh.se

Abstract

In this paper, we introduce the model REMES *for formal modeling and analysis of embedded resources such as storage, energy, communication, and computation. The model is a state-machine based behavioral language with support for hierarchical modeling, resource annotations, continuous time, and notions of explicit entry and exit points that make it suitable for component-based modeling of embedded systems.*

The analysis of REMES*-based systems is centered around a weighted sum in which the variables represent the amounts of consumed resources. We describe a number of important resource related analysis problems, including feasibility, trade-off, and optimal resource-utilization analysis. To formalize these problems and provide a basis for rigorous analysis, we show how to analyze* REMES *models using the framework of priced timed automata and weighted CTL. To illustrate the approach, we describe a case study in which it has been applied to model and analyze resource-usage of a temperature control system.*

1 Introduction

The importance of resource awareness in embedded systems is growing rapidly [6, 12, 13, 14, 18, 19, 21]. The limited availability of computing resources is preventing the introduction of new product features and applications, especially in areas where high-performance embedded systems are required. Resources include energy, computational power, memory, and hardware components such as buses, input/output ports, etc.

The systematic analysis of the resource consumption of an embedded system must include ways of semantic representation of various types of resources, be they of continuous type (like energy), or of discrete nature (e.g. memory, I/O ports). A representative analysis goal is to verify the *resource-wise feasibility* property. Such property can state that the composition of the worst-case resource requirements of components stays within the available resources provided by the implementation platform, or that there exists an execution path that uses no more than the available resources to behave correctly.

In practice, it may often be necessary to replace a component with another one having the same functionality, yet using a more sophisticated control algorithm that requires bigger memory resources. Alternatively, if one assumes a repository of models, the designer might need, at some point, to replace a component model with a refined one, having modified behavior or more efficiently implementable data structures.

We would be interested to assess, before deployment, how would any design decision, such as, reallocation of components to hardware units, or replacement of components, affect the system's overall resource consumption. This amounts to finding an appropriate trade-off between different configuration requirements and constraints.

The analysis of the embedded system's resource usage at an early design stage is extremely desirable. First, it allows for carrying out a potentially large number of design experiments, without increasing cost. Second, it may guide designers in making correct decisions, such as selecting the right components from a repository, or choosing among various admissible design models.

In this paper, we propose a modeling framework and apply associated analysis techniques for performing quantitative analysis such as feasibility, trade-offs, and optimal/worst-case resource consumption analysis. The model, called REMES, is tailored for embedded systems, but it is generally suitable for reactive systems. It provides support for discrete and continuous abstract resources characterized further by the way in which they are consumed and released, and by whether they can be referred to, or not. A number of generic resources can be modeled in this way, including memory, ports, energy, CPU, and buses. As such, the characterized and classified abstract resource types are not tied to any particular formal semantic interpretation.

In brief, our contribution is threefold:

- A classification of embedded resources, based on their rate of consumption over time, and the attribute of being referable, or not (section 3.1).

- The behavioral language REMES (sections 3.2, 3.3).
- Encoding the resource-wise analysis problem as a weighted sum of consumed resources (section 4.1).

Since REMES allows the description of functionality and timing in a dense time state-based modeling language, we also show how the latter and a number of important resource analysis problems can be formalized in the framework of (multi-)priced timed automata (sections 4.2, 4.3, 4.4).

The main purpose of REMES is to narrow the gap between architectural modeling (e.g., architecture description languages (ADLs) [20]) and formal analysis models (such as priced timed automata [2, 7]). This claim is supported throughout a case study presented in section 5, in which a temperature control system is modeled and analyzed. Following the example, we discuss our approach and compare to related work in section 6, then conclude the paper and present a line of future work, in section 7.

2 Preliminaries

2.1 Priced Timed Automata

In the following, we recall the model of priced (or weighted) timed automata [2, 7], an extension of timed automata [5] with prices/costs on both locations and transitions.

Let X be a finite set of clocks and $\mathcal{B}(X)$ the set of formulas obtained as conjunctions of atomic constraints of the form $x \bowtie n$, where $x \in X$, $n \in \mathbb{N}$, and $\bowtie \in \{<, \leq, =, \geq, >\}$. The elements of $\mathcal{B}(X)$ are called *clock constraints* over X.

Definition 1 *A linearly Priced Timed Automaton (PTA) over clocks X and actions Act is a tuple (L, l_0, E, I, P), where L is a finite set of locations, l_0 is the initial location, $E \subseteq L \times \mathcal{B}(X) \times Act \times \mathcal{P}(X) \times L$ is the set of edges, $I : L \to \mathcal{B}(X)$ assigns invariants to locations, and $P : (L \cup E) \to \mathbb{N}$ assigns prices (or costs) to both locations and edges. In the case of $(l, g, a, r, l') \in E$, we write $l \overset{g,a,r}{\to} l'$.* ■

The semantics of a PTA is defined in terms of a timed transition system over states of the form (l, u), where l is a location, $u \in \mathbf{R}^X$, and the initial state is (l_0, u_0), where u_0 assigned all clocks in X to 0. Intuitively, there are two kinds of transitions: delay transitions and discrete transitions. In delay transitions,

$$(l, u) \overset{d,p}{\to} (l, u \oplus d)$$

the assignment $u \oplus d$ is the result obtained by incrementing all clocks of the automata with the delay amount d, and $p = P(l) * d$ is the cost of performing the delay. Discrete transitions

$$(l, u) \overset{a,p}{\to} (l', u')$$

correspond to taking an edge $l \overset{g,a,r}{\to} l'$ for which the guard g is satisfied by u. The clock valuation u' of the target state is obtained by modifying u according to updates r. The cost $p = P((l, g, a, r, l'))$ is the price associated with the edge.

A timed trace σ of a PTA is a sequence of alternating delays and action transitions

$$\sigma = (l_0, u_0) \overset{a_1,p_1}{\to} (l_1, u_1) \overset{a_2,p_2}{\to} \dots \overset{a_n,p_n}{\to} (l_n, u_n)$$

and the cost of performing σ is $\sum_{i=1}^{n} p_i$. For a given state (l, u), the minimum cost of reaching (l, u) is the infimum of the costs of the finite traces ending in (l, u). Dually, the maximum cost of reaching (l, u) is the supremum of the costs of the finite traces ending in (l, u).

A network of PTA $A_1 || \dots || A_n$ over X and Act is defined as the parallel composition of n PTA over X and Act. Semantically, a network again describes a timed transition system obtained from those components, by requiring synchrony on delay transitions and requiring discrete transitions to synchronize on complementary actions (i.e., $a?$ is complementary to $a!$) [7].

In order to specify properties of PTA, the logic Weighted CTL (WCTL) has been introduced [11]. WCTL extends Timed CTL with resets and testing of cost variables. WCTL syntax is given by the following grammar:

$$\text{WCTL} \ni \phi ::= \mathbf{true} \mid a \mid \neg\phi \mid \phi \vee \phi \mid \mathbf{E}\,\phi\,\mathbf{U}_{\mathbf{P}\sim c}\,\phi \mid \mathbf{A}\,\phi\,\mathbf{U}_{\mathbf{P}\sim c}\,\phi$$

where a is an atomic proposition, $\mathbf{P}$ is a cost function, c ranges over $\mathbb{N}$, and $\sim \in \{<, \leq, =, \geq, >\}$. Here, **A**, and **E** are the universal and existential path quantifiers of WCTL, respectively, and $\mathbf{U}_{\mathbf{P}\sim c}$ is the "until" temporal modality. The temporal operators **F** and **G** are derived in the usual way. We interpret formulas of WCTL over labeled PTA, that is, PTA having a labeling function that associates with every location l a subset of atomic propositions. The satisfaction relation of WCTL is defined over configurations of the labeled PTA. We refer the reader to [11] for semantic details of WCTL.

2.2 Multi Priced Timed Automata

An extension of PTA is the class of Multi Priced Timed Automata (MPTA) in which a timed automaton is augmented with more than one cost variable [11, 17]. In the case of two costs associated with a PTA, the minimal cost reachability problem corresponds to finding a set of minimal cost pairs (p_1, p_2) (both p_1 and p_2 are minimized) reaching a goal state. Since the costs contributed from the individual costs can be incomparable, when, e.g. for the costs of two traces, say (p_1, p_2) and (p'_1, p'_2), $p'_1 < p_1$ and $p_2 < p'_2$, the solution is a set of pairs, rather than a single pair. In this setting, the minimal cost reachability problem is to find the set of incomparable pairs with minimum cost, reaching the goal state. Dually, the maximization problem is defined as finding the set of incomparable pairs with

Resource Class	Characteristics
A (memory)	discrete: $\dot{c} = 0$ referable
B (CPU, bandwidth)	discrete: $\dot{c} = 0$ non-referable
C (CPU, energy, bandwidth)	continuous: $\dot{c} = n$, $n \in \mathbb{Z}$ non-referable

Table 1. Resource classes/characteristics

maximal cost reaching the target location, or to conclude (∞, ∞) if the target location is avoidable in a path that is infinite, deadlocked, or has a location in which it can make an infinite delay. A specific problem is the optimal conditional reachability problem, in which one of the costs should be optimized, and the other bounded by an upper/lower bound. We refer the reader to [17] for a thorough description of optimization problems in MPTA.

3 REMES: The Proposed Resource Model

In this section, we define the resources of interest and introduce the model REMES intended for resource modeling and analysis.

3.1 Classes of resources

We consider resources as global quantities of finite size. We refer to the *consumption* of a resource c as being the accumulated resource usage up to some point in time, whereas the derivative of c, denoted $\dot{c}$, is the rate of consumption over time. Resource consumption can be of *discrete* or *continuous* nature. We also classify resources depending on whether they are *referable* or *non-referable*. A representative example of a referable resource is the memory. Memory can be dynamically allocated, deallocated, addressed, and manipulated during run-time.

Taking all these into consideration, Table 1 shows three identified resource classes and their characteristics of interest. Resource consumption for resources that belong to class C is continuous, which is in opposition to the discrete resource consumption nature for the resources from class A and B. The consumption of the CPU can be modeled either by a discrete variable, denoting the number of accumulated clock ticks, or by a continuous variable, which encodes the processor load (that is, the derivative describes, e.g., how many tasks are starting execution, every time unit). Accordingly, CPU may be in either class B or C (same applies to bandwidth). Only the resources from class A are referable and can be dynamically manipulated.

3.2 Introducing REMES

Our **RE**source **M**odel for **E**mbedded **S**ystems (REMES) is intended to describe the resource-wise behavior of interacting embedded components. REMES relies heavily on the modeling language CHARON [4], used for specifying embedded systems as communicating agents. Our main contribution is the addition of information regarding resource consumption and its rate, as well as other constructs that facilitate the application of REMES to modeling both functional and extra-functional behavior of (real-time) component-based systems.

In REMES, the internal behavior of a component is described by a *mode*. We call a mode *atomic* if it does not contain any submode, and *composite* if it contains a number of submodes (see Figure 1). Like in CHARON, the data is transferred between modes via a well-defined *data interface*, that is, typed global variables, whereas the (discrete) control is passed through a well-defined *control interface* consisting of *entry* and *exit* points. Observe, in Figure 1, that the entry and exit points are drawn as blank and filled circles, respectively. The variables of mode M are partitioned into *local* variables, ($\mathsf{L_M}$), and *global* variables ($\mathsf{G_M}$), and can be of types boolean, natural, integer, array, or of an extra type clock that specifies continues variables evolving at rate 1. The global variables are in turn partitioned into *read* variables, $\mathsf{Rd_M}$, and *write* variables, $\mathsf{Wr_M}$, such that $\mathsf{G_M} = \mathsf{Rd_M} \cup \mathsf{Wr_M}$.

Read/Write Variable Accesses. The local variables of M, $\mathsf{L_M}$, can not be read or written by other modes, the set $\mathsf{Wr_M}$, written by M can be read by other modes, whereas the set $\mathsf{Rd_M}$ may be written by other modes. The sets $\mathsf{Wr_M}$ and $\mathsf{Rd_M}$ need not be disjoint; concurrent access to common write variables of modes can be regulated by specifying certain synchronization protocols in the REMES model.

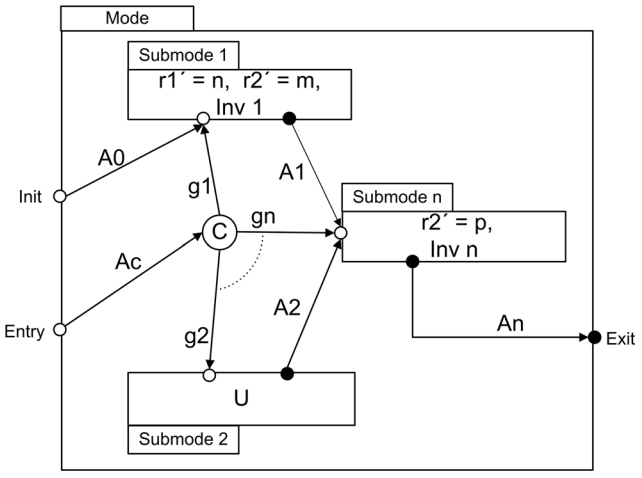


Figure 1. A REMES Composite Mode.

The atomic modes Submode 1 and Submode n in Figure 1 are annotated with their respective resource-wise continuous behavior, assuming that the corresponding component is consuming resources $(r1, r2)$ belonging to class C. Such consumption is expressed by the first derivatives of the typed resource variables $r1, r2 : T_C$, respectively, $r1', r2'$, which give the rates at which the composite mode consumes the resources in time, depending on the executing submode.

For a composite mode, the control flow is given by a set of directed lines, called *edges*, which connect the control points of the submodes, or of the composite mode and its submodes. For example, in Figure 1, the composite mode takes the edge labeled A_0, in order to enter Submode 1, after initialization, and similarly, edges labeled $A_1, A_2, \ldots, A_n$, to further enter Submode 1, Submode 2, ..., Submode n, respectively.

REMES supports two types of actions, *delay/timed* actions and *discrete* actions. A delay action describes the continuous behavior of the mode, and its execution does not change the mode. The delay/timed actions are not visible in a REMES model, but they are usually constrained by the differential equations that annotate the modes, and they represent the solutions of such equations. Observe that Submode 2 is decorated with letter U, meaning that such a mode exits its right-away after its activation, without any delay. Such modes are called *urgent*. On the other hand, discrete actions are instantaneous actions, and they are represented as annotations of the edges. Executing a discrete action results in a mode change, by taking the outgoing edge starting from the mode's exit point.

A discrete action $A = (g, S)$ is a statement list prefixed by a boolean expression, with g called the *action guard*, and S the *action body*, that is, the statement (assignment, conditional statement etc.) or sequence of statements that must be executed once the corresponding edge has been taken. We say that a discrete action A is *enabled*, hence it could be executed, if its corresponding guard g evaluates to TRUE at some point in time. A discrete action is called *always enabled* if its guard always holds, and *empty* if its body does not change any of the mode variables (in such cases, the action body can be omitted).

In addition, one needs to specify for how long a (sub)mode is executed, so an *invariant*, e.g., Inv 1,..., Inv n, that is, a predicate over continuous variables, captures such a timing constraint. Once the invariant stops to hold, the mode is exited by taking one of the outgoing edges.

Similar to Statecharts [15], REMES provides a *conditional connector* (depicted by C in Figure 1), which allows the selection of an outgoing edge, out of two or more possible ones, via the guarding boolean conditions (guards $g_1, g_2, \ldots, g_n$) of the discrete actions that correspond to the edges exiting the conditional connector. For a discrete action to be possibly executed, the component must be in the right mode and the corresponding guard must evaluate to TRUE. If none of the guards evaluates to TRUE, then no discrete action is executed and the component remains in its current mode, performing delay actions. If more than one action guards are TRUE, then one of the enabled discrete actions could be executed non-deterministically.

We classify a mode's edges, and afferent discrete actions, as follows:

- *entry edges*: connect an entry point of the composite mode with the entry point of a submode (e.g., annotated with action A_0);
- *exit edges*: connect the exit point of a submode with the exit point of the composite mode (e.g., annotated with action A_n);
- *entry conditional top edges*: connect an entry point of the composite mode with a conditional connector (e.g., annotated with action A_C);
- *exit conditional top edges*: connect a conditional connector with the exit point of the composite mode;
- *entry conditional sub edges*: connect a conditional connector with the entry point of a submode;
- *exit conditional sub edges*: connect the exit point of a submode with a conditional connector;
- *internal edges*: connect the exit point of a submode with the entry point of another submode (e.g., annotated with actions A_1, A_2).

The control points of submodes should be connected by edges, such that the termination of the internal behavior of the composite mode is ensured (e.g., in case cases exist, termination can be guaranteed by decreasing a chosen termination function each time the cycle is executed). Note that, in REMES, each mode describes the behavior of a component, and a composite mode is a way of encapsulating behavior and it describes a composite component.

Formal Definition of a Mode. A mode M is a tuple $(\mathsf{SM}, \mathsf{V}, \mathsf{In}, \mathsf{Out}, \mathsf{E}, \mathsf{RC}, \mathsf{Inv})$, where: SM is the set of submodes, V is the set of variables, In is the set of entry control points, Out is the set of exit control points, E, the set of edges, RC, the set of resource constraints that define the admissible values for the consumption rates of the involved resources in class C, and, finally, Inv is the set of invariants.

For the submodes of M, the following condition should hold: $\mathsf{G_{SM}} \subseteq \mathsf{L_M} \cup \mathsf{G_M}$, for a local variable of a mode to be accessible only in its submodes, and not anywhere else.

Mode Execution. The top-level mode, which is activated when a corresponding event is received, enters execution for the first time through the special Init entry point, while initializing the global variables, accordingly. After that, the mode is re-entered through control point Entry.

A mode can execute either a *discrete step*, by a discrete action, or a *continuous step*, via a delay action, with such steps alternating as dictated by the urgency of the mode. When executing a continuous step, the mode follows a continuous path that satisfies the resource constraints (RC). When the mode invariant is violated, the mode must execute an outgoing discrete step. A discrete step of a mode is a finite sequence of discrete steps of the submodes, that is, a sequence of executing discrete actions. A discrete step begins in the current mode and ends either at the entry point of a submode, or when it reaches the current mode's exit point, meaning that the current mode has passed control to some other mode.

The fact that a mode can pass control is ensured by the *closure* construction: each exit point of a mode is either connected to the exit point of the composite mode, or deterministically connected to an entry point of another mode that eventually leads to the composite mode's exit.

For example, in Figure 1, the execution of Mode proceeds as follows: after initialization, the discrete step corresponding to A_0 is executed, after which a sequence of continuous steps is executed, until the invariant Inv 1 fails to hold; alternatively, in case A_1's guard evaluates to TRUE, the mode could take a discrete step and entry Submode n. Next, a similar sequence follows, while the mode executes Submode n. When Inv n does not hold anymore, the mode takes a new discrete step corresponding to discrete action A_n. The next time when the control is passed to Mode, a discrete step corresponding to A_C is taken and the selection of a possible path is made through the conditional connector, etc.

3.3 Composition of REMES models

REMES atomic modes and composite modes can be composed in parallel with each other. The parallel modes can execute concurrently, by interleaving actions, whereas the submodes can never execute in parallel; they simply obey the strict execution order imposed by the control flow.

Like in CHARON, if M is a composite mode, and $\mathsf{sm} \in \mathsf{M}$ is the variable ranging over the constituent submodes, we have: $\mathsf{L_M} \subseteq \cup_{\mathsf{sm}\in\mathsf{M}}\mathsf{G_{sm}}$, and $\mathsf{G_M} = \cup_{\mathsf{sm}\in\mathsf{M}}\mathsf{G_{sm}} - \mathsf{L_M}$. The mode composition is defined as follows.

Definition 2 *Assume* $\mathsf{Mode_A}$ *and* $\mathsf{Mode_B}$ *are two* REMES *(atomic or composite) modes. Then, the composition* $\mathsf{Mode_D} = \mathsf{Mode_A} \,||\, \mathsf{Mode_B}$ *is the mode with the set of local variables* $\mathsf{L_{Mode_D}} = \mathsf{L_{Mode_A}} \cup \mathsf{L_{Mode_B}}$*, the set of write variables* $\mathsf{Wr_{Mode_D}} = \mathsf{Wr_{Mode_A}} \cup \mathsf{Wr_{Mode_B}}$*, the set of read variables* $\mathsf{Rd_{Mode_D}} = \mathsf{Rd_{Mode_A}} \cup \mathsf{Rd_{Mode_B}}$*, and the top-level mode* $\mathsf{Mode_A} \cup \mathsf{Mode_B}$. ■

In Definition 2, the parallel composition of composite modes subsumes the reunion of all the constituent submodes, corresponding edges and associated actions.

4 Analyzing REMES-based Systems

4.1 Analysis model for REMES

Assume a set of resources $R_1, \ldots, R_n$ that a set of REMES modes have access to. Our main goal is to analyze various scenarios of the system's resource usage, and be able to compute, e.g., the maximum or minimum amounts of needed resources for guaranteeing correct resource-wise system behavior. Intuitively, this problem reduces to a scalar problem if one constructs a weighted sum of all resource consumptions, which should then be minimized, maximized, or manipulated in order to compute trade-offs. Consequently, we propose the following function as the analysis model for REMES:

$$r_{tot} \triangleq w_1 \times r_1 + w_2 \times r_2 + \ldots + w_n \times r_n,$$

where variable r_{tot} represents the total consumption of resources $R_1, \ldots, R_n$, and variables $r_1, \ldots, r_n$ denote the accumulated consumption of $R_1, \ldots, R_n$, respectively. The constants, $w_1, \ldots, w_n$ (weights), represent the relative importance of $r_1, \ldots, r_n$. The values of the weights are a subjective matter; the way they are chosen depends both on the application and on the analysis goals. For example, in designing a heavily resource-constrained soft real-time embedded system that might tolerate lateness at the expense of quality of service, in order to determine trade-offs between memory consumption and (execution) time, one can assign higher weight to memory than to time.

Informal Translation of REMES into PTA. In order to be able to analyze REMES compositions, formally, we need a semantic translation of the model. If we consider resource consumptions $r_1, \ldots, r_n$ as cost variables $c_1, \ldots, c_n$, we can use the framework of Priced Timed Automata as the underlying semantic representation.

Informally, translating REMES into PTA is quite straightforward: the syntactic REMES element of an edge corresponds to an edge in PTA, whereas the REMES semantic discrete step is a transition in PTA's semantics. An atomic mode represents a PTA location, and global variables used for passing control in REMES become synchronization channels in PTA. The formal translation of the hierarchical REMES into a network of PTA and the associated tool are subject to future work. Next, we formalize some of the main analysis goals that we are interested in.

4.2 Feasibility Analysis

Component-based feasibility analysis reduces to checking whether the accumulated values of the resources consumed/used during all possible system behaviors are within the available resource amounts provided by the implementation platform. For resources like non-referable memory and

energy, the composition of individual resource consumptions of REMES components is additive.

If one considers the PTA model of Definition 1 as the semantic translation of a REMES model, feasibility goals can then be formalized as the following WCTL properties that the PTA model can be checked against:

$$A\,F_{cost\leq n}\,v \tag{1}$$
$$A\,G\,(q \Rightarrow A\,F_{cost\leq n}\,v) \tag{2}$$
$$E\,F_{cost\leq n}\,v \tag{3}$$
$$A\,G\,(q \Rightarrow E\,F_{cost\leq n}\,v) \tag{4}$$

where G and F are the WCTL temporal operators "always" and "eventually", respectively [11].

The above properties are in fact liveness properties (1), (2), (4), and a reachability property (3), indexed by cost constraints. The first two properties specify *strong feasibility*: property (1) requires that for all execution paths, the target location v is eventually reached within a total cost of n that can model the available resources provided by the platform; property (2) states that, for all paths, it is always the case that, once q, the cost of eventually reaching v will be no more than n, regardless of how v is reached. We say that property (3) models *weak feasibility*: the target location v may be reached within a total cost of n. Finally, property (4) states that for all paths, it is always the case that once a location q is reached, there exists a way by which v will be eventually reached within cost n. We call this last property *live feasibility*. However, model-checking WCTL formulae is decidable just for one-clock priced automata [10]. For other PTA, one can only verify reachability properties of the form given by (3).

Assuming that the cost function equates to $cost = w_1 \times c_1 + \ldots + w_n \times c_n$, and $c_1, \ldots, c_n$ are constants, the feasibility checks of the above properties involve a single cost variable that represents the accumulated resource consumption of all resources of interest, regardless of the class they belong to. Hence, semantically, the various resources become undistinguishable in these cases.

4.3 Optimal and Worst-Case Resource Consumption

Optimal and worst-case resource consumption analyses require (symbolic) algorithms on PTA, which compute the cost of the "cheapest", and/or most "expensive" trace that will eventually reach some goal. The optimal/worst-case resource consumption problem reduces to minimizing/maximizing the one-cost function $cost = w_1 \times c_1 + \ldots + w_n \times c_n$, such that a given reachability, or liveness property is satisfied.

Finding the optimal/worst-case resource consumption values to attain such goals calls for synthesis algorithms of minimal/maximal reachability costs for PTA, which have been proposed by Larsen and Rasmussen [17]. Similar to the feasibility case, only optimal/worst-case reachability costs can be synthesized by a model-checker. Later, we show how such a cost-optimal trace can be actually computed in the example of section 5.

A considerable verification challenge arises in case some of the edge prices are negative, so that $cost$ becomes a non-monotonically increasing cost function. In such situations, the usual branch-and-bound symbolic reachability algorithms, for PTA, cannot be applied as such anymore, since minimal/maximal reachability analysis requires a monotonically increasing cost function. The optimal- and worst-case-cost reachability problems have been theoretically solved even when negative costs are involved [9].

The tool used for verifying optimal resource consumption properties is UPPAAL CORA, where one could check, e.g., the relevant reachability property, $E\,F\,v$, while the tool calculates the minimum cost, in terms of resource exemption, "paid" to satisfy the property.

4.4 Trade-off Analysis

Minimization of memory usage plays a major role in the design of embedded systems. Limited memory is one of the dominating constraints for many advanced embedded systems. However, while trying to minimize memory consumption, one might be forced to increase the execution time of real-time components beyond acceptable limits, that is, limits that, if exceeded, would make the set unschedulable.

As such, for a given REMES model, we may have more than one property to satisfy simultaneously, and we want to know whether it is possible to satisfy all of them, although they might be subjected to apparently conflicting constraints. In such cases, there should be possible to compute a *trade-off* between the considered resource consumptions.

Computing a trade-off between memory and execution time, or between any resource belonging to classes A and B, or A and C, or B and C of Table 1, could be done in PTA, by employing a single-cost function. The trade-off could then be achieved by varying the weights $w_1, \ldots, w_n$, accordingly.

In some other cases, e.g., when one needs to compute trade-offs between consumption of resources belonging to class C, the function $cost = w_1 \times c_1 + \ldots + w_n \times c_n$ becomes a multi-cost function that lets one distinguish between various types of resources (e.g., between energy and CPU). This forces one to carry out the analysis on MPTA, rather than on PTA.

Assuming energy and CPU as the resources of interest, we want to determine which are the simultaneously achiev-

able pairs of costs $(w_{eng} \times c_{eng}, w_{cpu} \times c_{cpu})$ such that energy consumption is minimized, while CPU consumption remains bounded from above.Such synthesis of cost pairs, which can be seen as a variant of trade-off analysis, can be achieved by applying *optimal conditional reachability* algorithms on MPTA [16], while considering c_{eng} as the primary cost and c_{cpu} as the secondary cost. Larsen and Rasmussen have proved that such problems are decidable for MPTA [16].

Alternatively, one could perform a feasibility-like check, by requiring that the following WCTL property is satisfied:

$$E\, F_{(w_{eng} \times c_{eng}) \leq n}\, (v \wedge (w_{cpu} \times c_{cpu}) \leq m)$$

The formula states that the accumulated weighted CPU usage will not be more than m ticks at location v, while v may be reached by consuming no more than n weighted energy units.

5 Example: A Temperature Control System

As a case study (taken from [3]) demonstrating the principles of our resource modeling and analysis approach, we consider a temperature control system (TCS) for a heat producing reactor. It has two rods that can be inserted into the core of the reactor, to control the heat producing (chain) reaction. If inserted into the core, the control roads absorb neutrons and consequently the reaction is slowed down, so the temperature inside the core starts decreasing. If they are pulled out, the reaction speeds up again, which in turn increases the core temperature. The goal of the TCS is to maintain the temperature in the reactor core between θ_{min} and θ_{max}. Whenever the core reaches temperature θ_{max}, it has to be cooled with one of the two rods. After a rod has been used for cooling, it is then unavailable for T time units.

5.1 A REMES Model of TCS

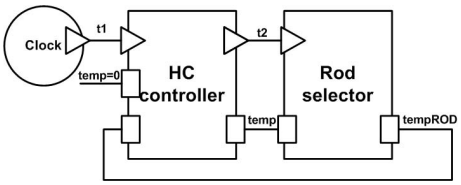


Figure 2. Component based TCS model.

We model an abstracted version of the internal design of TCS in the SaveComp component model [1], with three components: HC controller, Rod selector, and Clock as depicted in Figure 2. The interfaces of the components are described in terms of ports. SaveComp distinguishes between input and output ports, which can be of the types: *data* for transferring of data, *triggering* to trigger component executions, or *combined* to combine the two.

The component HC controller activates the heating/cooling process of the core using trigger port t2. The

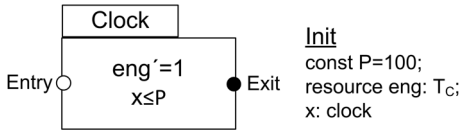


Figure 3. The Clock modeled in REMES.

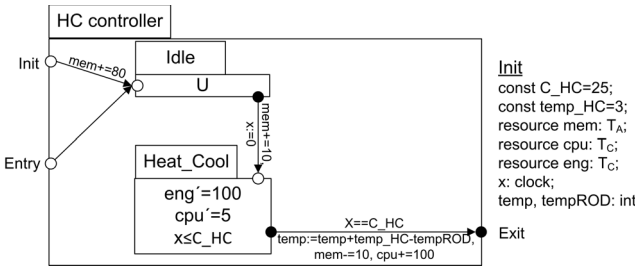
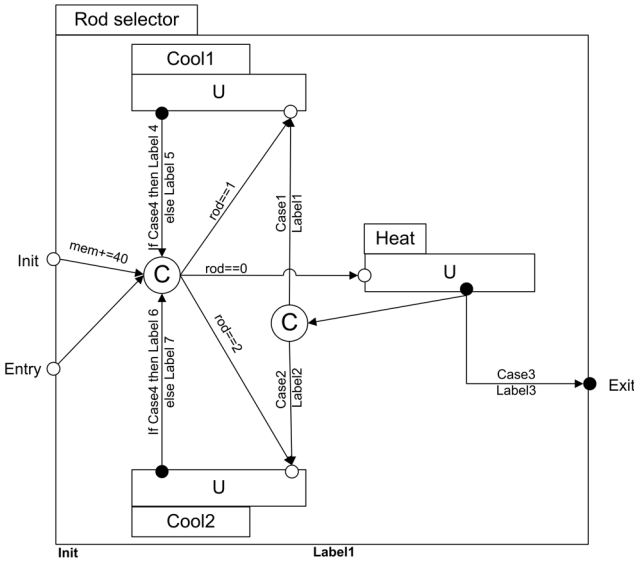


Figure 4. The HC Controller modeled in REMES.



Init
const T=3, theta_max=25, theta_min=6;
const margin=5; R1=5, R2=6;
resource mem: T_A;
resource cpu: T_C;
resource eng: Tc;
x: clock;
temp, tempROD, x1, x2, rod: int
Case1
x1>=T&&temp>=(theta_max-margin))
Case2
x2>=T&&temp>=(theta_max-margin))
Case3=not(Case1 or Case2)
Case4
temp>(theta_min+margin)

Label1
rod:=1, cpu+=2, mem+=2
Label2
rod:=2, cpu+=1, mem+=7
Label3
x1:=(x1==T?x1:x1+1), x2:=(x2==T?x2:x2+1),
cpu+=10
Label4
tempROD:=R1, cpu+=1
Label5
tempROD:=0, rod:=0, x1:=0, cpu+=3, mem-=2
Label6
tempROD:=R2, cpu+=1
Label7
tempROD:=0, rod:=0, x2:=0, cpu+=3, mem-=7

Figure 5. The Rod selector modeled in REMES.

Rod selector uses temperature data of the core conveyed through data port temp to control whether the core should continue to heat, or if a rod should be selected for insertion into the core to slow down the reaction. The latter must take the availability of rods into consideration, as a rod has to rest for at least T time units after its previous use. Finally, the Clock component, periodically generates the trigger event t1 that activates the HC controller. The temp value in the HC controller is updated by reading the value of variable tempROD that is assigned the cooling rate of the rods within the Rod selector component.

We model the resource usage of the TCS components as modes in REMES. The modes of the Clock, the HC controller, and the Rod selector are depicted in Figure 3, 4, and 5, respectively. The modes communicate data between each other using the global variables: temp, tempROD, t1, and t2. The modes of HC controller and the Rod selector are made of submodes, conditional connectors, and edges, as described in Section 3.

In the TCS model, we make use of three resources: memory, energy, and CPU, which belong to two different classes of the taxonomy presented in Section 3.1. We assume that every simple cpu instruction utilizes one cpu tick. We treat static memory and simple dynamic memory that is allocated when a mode is entered and released as soon as the same mode is exited, without memory management.

5.2 A PTA model of TCS

We have analyzed the REMES-based TCS system, as a network of three PTA models, in UPPAAL CORA[1]. The PTA models of the Clock, the HC controller, and the Rod selector are shown in Figure 6.

The Clock is modeled as a simple PTA that, after every P time units, periodically synchronizes on channel t1 with HC controller. The HC controller PTA has three locations: Start, Idle, and Heat_Cool. The constant C_HC is the execution time of the HC controller. The difference $(\mathsf{temp\_HC} - \mathsf{tempROD})$, where temp_HC is the heating produced by the reactor, and tempROD is the current cooling effect of the rod, is used to update the reactor temperature.

The PTA Rod selector has five locations: Start, Select, Heat, Cool1, and Cool2. The execution of the Rod selector consumes 40 units of static memory. The locations Start, Heat, Cool1, and Cool2 are committed, as their actions are atomic. The synchronization with HC controller is modeled using channel t2. The selection of the rods is controlled by variable rod. From location Heat, based on the temperature of the core, temp, and the time since a rod has been previously used for cooling (i.e., x1 and x2 for rod1 and rod2, respectively), an available rod is selected for insertion into the core, and, consequently, the Rod selector enters location Cool1 or Cool2, or alternatively jumps back to location Select, provided that no rod needs to be used.

For analysis purposes, we have added the TCS model with the function run() (see Figure 6(c)) that merely stores the first few selections of rods, in an array of integers.

5.3 Formal Analysis of the PTA model

In the analysis model, we have encoded the relative importance of the resources energy, CPU, and memory. We consider CPU to be the most critical resource, followed by memory. Energy is not critical, yet it is taken into consideration in order to ensure higher energy efficiency in the system. Therefore, we give highest weight to CPU and lowest to energy. The cost of resource usage is influenced by the individual weights of each resource, and the consumed (utilized) resource on each transition (location). Currently UPPAAL CORA can only handle PTA models where the cost function is monotonically increasing. This means that in order to keep the cost function monotonically increasing we have to fine-tune the weights of the resources.

In the TCS system, we consider the following total cost function

$$c_{tot} = \mathsf{wcpu} \times c_{cpu} + \mathsf{wmem} \times c_{mem} + \mathsf{weng} \times c_{eng}$$

where wcpu = 15, wmem = 2, and weng = 1, and c_{cpu}, c_{mem}, and c_{eng} are the accumulated consumed amounts of cpu, memory, and energy, respectively.

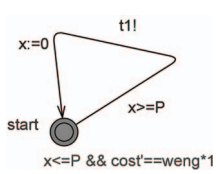
After having fed UPPAAL CORA with the PTA model of the TCS, we were able to analyze the minimum cost reachability problem, that is, to compute the lowest cost of satisfying a given reachability property, and a corresponding trace. However, we have first checked the model against the safety properties: $\mathsf{AG\,temp} \leq \mathsf{theta_{max}}$ and $\mathsf{AG\,temp} \geq \mathsf{theta_{min}}$.

In our case, we are interested in finding an execution order of the system (a cheapest sequence of rod insertions) that results in the lowest possible total resource cost, that is, to minimize c_{tot}. Such information extracted from the analysis could be used in the implementation stages of the TCS system, by resolving existing non-determinism in such a way that a specific execution trace, the cheapest with respect to total resource usage, is enforced.

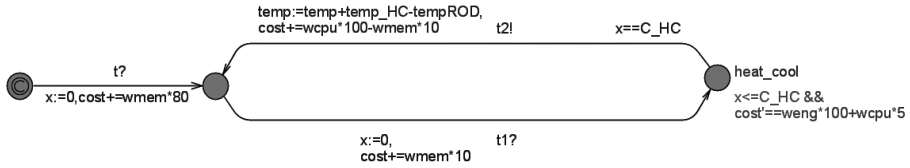
To illustrate the technique, we check for an optimal trace satisfying the property: $\mathsf{EF}(\mathsf{count} == 3)$, that is, a trace in which rods are inserted into the reactor three times. UPPAAL CORA has found that the second rod should be inserted two times, followed by the first one, the third time. Table 2 shows the cost of this best trace, and also the cost of another more expensive trace where only rod 2 has been used.

For TCS, we can only partially tackle the trade-off resource analysis problem, by giving higher weight to the

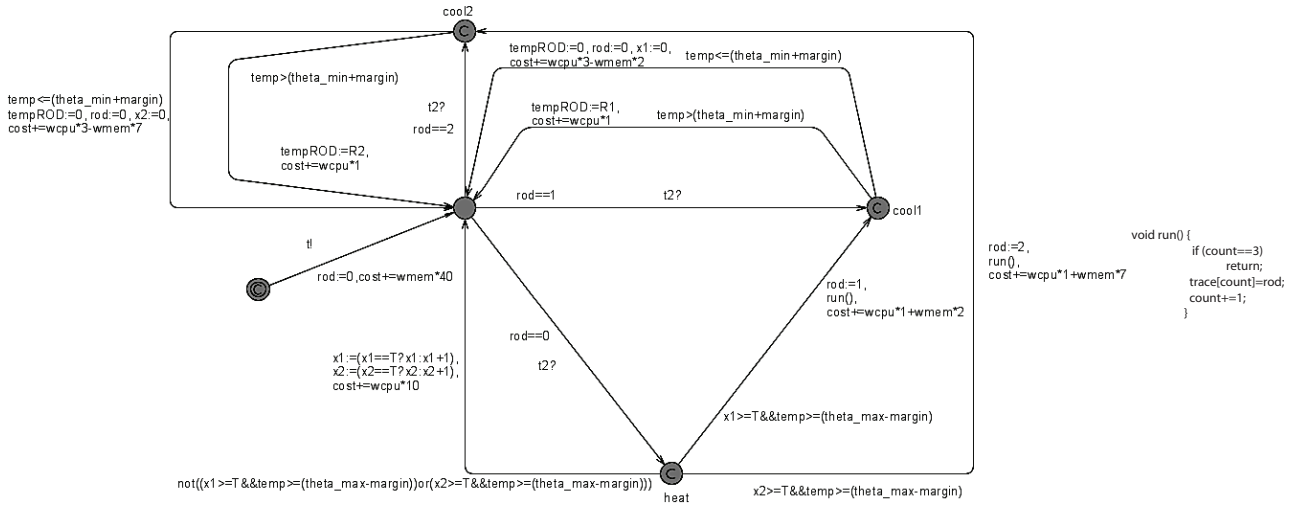
[1] See the web page www.uppaal.org/cora for more information about the UPPAAL CORA tool.



(a) The model of the clock component as a PTA.



(b) The model of the HC controller component as a PTA.



(c) The model of the Rod selector component as a PTA.

Figure 6. TCS modeled with three PTA.

most critical resource, the CPU, followed by memory and energy. Additionally, we have conducted optimal reachability resource usage analysis, by minimizing the memory consumption, while imposing upper bounds on the CPU consumption, in the TCS. For example, for three sequential insertions of the rods in the reactor's core, it might happen that it is necessary to insert the second rod three times in a row, in order to satisfy all constraints, even though the total cost is higher for such a trace than for the best execution trace.

Scenario	Order of execution	Cost
1	P_2-P_2-P_1	127499
2	P_2-P_2-P_2	127509

Table 2. Cost of execution for different rod insertion scenarios.

6 Discussion and Related Work

The REMES model presented in Section 3 can be employed in the design of embedded systems, for representing the internal behavior of the interacting embedded components. As such, it complements architectural description languages (ADLs) [20], which describe the software system's conceptual architecture as a collection of components, connectors and architectural configurations, by adding component behavior. If one attaches semantics to the connection points of the architectural elements of a system, REMES can then be used for modeling the behavior of a generic embedded system. Moreover, we believe that REMES is simple enough to be utilized by both formalists and engineers with different backgrounds, as an intermediate layer between abstract architectural modeling and very detailed behavioral modeling (e.g., by PTA [2, 7]).

The cost analysis model proposed in Section 4.1 is platform-aware. Hence, as future work, it could benefit from including abstractions of platform specific tools, such as the associated compiler, linker etc. We do believe that the cost model can be derived from the results provided by static analysis tools, which could be applied on already implemented components. A possible solution is presented by Bonenfant et al. [8]. In order to obtain provably correct static analysis results, the authors propose a formal source-level cost model, enriched with rules for deriving the execu-

tion cost of a subset of expressions belonging to the system-oriented language Hume.

Last but not least, we underline the fact that the selection of the weights in our resource model depends mostly on the designer's experience and decisions. However, by analyzing the results of model checking the chosen cost models, one could adjust the weights accordingly.

Related Work. In a recent study [23], Vulgarakis and Seceleanu have presented related work on modeling and analyzing resources in component-based embedded real-time systems, and they have grouped it into three categories, as follows.

First, research has been devoted to code-level resource modeling and analysis, in component assemblies. In Koala [13] and Robocop [12] component frameworks, static memory estimation has been performed for applications in which the instantiated components of a composition are known prior to run-time. Such low-level code-driven resource estimates can only be used in cases when one has access to the components implementations. More abstract descriptions of expected resource usage may be needed for not-yet implemented components, or for guiding the selection of components from the repository. In such cases, the designer could first employ REMES for early resource usage analysis, and then apply the approaches mentioned above.

The second category is represented by the UML-based attempts [14, 6] that have been undertaken to tackle the analysis of embedded resources. Although graphical and intuitive, these approaches lack a formal description that could provide the designer with verified resource usage claims. In contrast, REMES provides both a graphical behavioral notation, as well as a rigorous underlying framework for formal analysis.

Third, higher-level formal approaches [18, 19], proposed by Lee et al., encompass a family of process-algebraic formalisms, developed to unify formal modeling and analysis of embedded systems resources. The framework is theoretically rich, yet the tool support is not equally mature. Ouimet et al. use timed abstract state machines [21] to describe resources as simple annotations, in the form of real-valued variable assignments. Consequently, the framework can not support trade-off analysis of possibly conflicting resource requirements.

Last but not least, as mentioned earlier, REMES focuses on component-based behavioral modeling, and, if paired with ADL descriptions [20], could provide the designer with a complete system representation.

7 Conclusions and Future Work

In this paper, we have introduced REMES — a language for resource modeling and analysis of embedded systems. The essence of REMES is a notion of resources that are characterized by their discrete or continuous nature, the way they are consumed and/or allocated and released, and whether they can be referred to, or not. Resources that can be easily modeled include memory, ports, energy, CPU, busses etc.

In order to express resource usage in a system, REMES has a graphical behavioral language influenced by CHARON, timed and hybrid automata, and Statecharts. The language supports hierarchical modeling and has notions of explicit entry and exit points that make it suitable as a semantic basis in component based development frameworks. REMES has notions of continuous variables, flows, and progress constraints (invariants), which fit modeling timed behaviors in embedded systems.

In this setting, we have defined three important resource analysis problems: feasibility analysis, trade-off analysis, and optimal/worst-case resource analysis. All these problems rely on weighted sums of consumed amounts of resources and their given weights. In this way, the analysis can result in optimizing the overall resource usage of a system, with respect to parameters such as criticality or costs of the available resources.

To illustrate analysis, we have shown in an example how REMES models can be analyzed in the framework of (multi) priced timed automata. The studied example is a temperature control system of a reactor that consumes CPU, energy, and memory resources. The system is architecturally modeled in the component modeling language SaveCCM, and REMES is used to describe function, timing, and resource usage of the included components. To synthesize the optimal resource usage of the system, we model the latter and the weighted sum of resource costs, as a network of PTA, and perform the analysis in the UPPAAL CORA tool.

As future work, we plan to apply the results of Bouyer et al. [9], in order to tackle the feasibility analysis problem for systems in which the global cost function is non-monotonic. We also plan to integrate REMES and its notion of resources in the recently proposed ProCom component model [22] and its associated tools.

Acknowledgments: This work was partially supported by the Swedish Foundation for Strategic Research via the strategic research centre PROGRESS, the Swedish Research Council (VR), and the European Union via the Q-ImPrESS research project (FP7-215013).

References

[1] M. Åkerholm, J. Carlson, J. Fredriksson, H. Hansson, J. Håkansson, A. Möller, P. Pettersson, and M. Tivoli. The save approach to component-based development of vehicular systems. *Journal of Systems and Software*, 80(5):655–667, May 2007.

[2] R. Alur. Optimal paths in weighted timed automata. In *In HSCC01: Hybrid Systems: Computation and Control*, pages 49–62. Springer, 2001.

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

[4] R. Alur, T. Dang, J. Esposito, Y. Hur, F. Ivančić, V. Kumar, I. Lee, P. Mishra, G. Pappas, and O. Sokolsky. Hierarchical modeling and analysis of embedded systems. *Proceedings of the IEEE*, 8(3):231–274, 1987.

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

[6] H. H. Ammar, V. Cortellessa, and A. Ibrahim. Modeling resources in a uml-based simulative environment. In *AICCSA*, pages 405–410, 2001.

[7] G. Behrmann, A. Fehnker, T. Hune, K. G. Larsen, P. Pettersson, J. Romijn, and F. Vaandrager. Minimum-Cost Reachability for Priced Timed Automata. In M. D. D. Benedetto and A. Sangiovanni-Vincentelli, editors, *Proceedings of the 4th International Workshop on Hybris Systems: Computation and Control*, number 2034 in Lecture Notes in Computer Sciences, pages 147–161. Springer–Verlag, 2001.

[8] A. Bonenfant, Z. Chen, K. Hammond, G. Michaelson, A. Wallace, and I. Wallace. Towards resource-certified software: A formal cost model for time and its application to an image-processing example. In *ACM Symposium on Applied Computing (SAC '07), Seoul, Korea, March 11-15*, 2007.

[9] P. Bouyer, T. Brihaye, V. Bruyère, and J.-F. Raskin. On the optimal reachability problem. *Formal Methods in System Design*, 31(2):135–175, 2007.

[10] P. Bouyer, K. G. Larsen, and N. Markey. Model-checking one-clock priced timed automata. *Logical Methods in Computer Science*, 4(2:9):1–28, 2008.

[11] T. Brihaye, V. Bruyère, and J.-F. Raskin. Model-checking for weighted timed automata. In *Proc. of FORMATS-FTRTFT04*, number 3253 in Lecture Notes in Computer Science, pages 277–292. Springer–Verlag, 2004.

[12] M. de Jonge, J. Muskens, and M. Chaudron. Scenario-based prediction of run-time resource consumption in component-based software systems. In *Proceedings of the 6th ICSE Workshop on Component-based Software Engineering (CBSE6)*, pages 19–24. IEEE, 2003.

[13] A. V. Fioukov, E. M. Eskenazi, D. K. Hammer, and M. R. V. Chaudron. Evaluation of static properties for component-based architectures. In *EUROMICRO*, pages 33–39, 2002.

[14] O. M. Group. Uml profile for schedulability, perfomance and time specification. *Version 1.1, formal / 05-01-02*, 2005.

[15] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 91(1), 2003.

[16] K. G. Larsen and J. I. Rasmussen. Optimal conditional reachability for multi-priced timed automata. In *Proceedings of the 8th International Conference on Foundations of Software Science and Computational Structures (FOSSACS 2005/ETAPS 2005)*, number 3441 in Lecture Notes in Computer Sciences, pages 234–249. Springer–Verlag, 2005.

[17] K. G. Larsen and J. I. Rasmussen. Optimal reachability for multi-priced timed automata. *Theor. Comput. Sci.*, 390(2-3):197–213, 2008.

[18] I. Lee, A. Philippou, and O. Sokolsky. A general resource framework for real-time systems. In *RISSEF*, pages 234–248, 2002.

[19] I. Lee, A. Philippou, and O. Sokolsky. Resources in process algebra. *J. Log. Algebr. Program.*, 72(1):98–122, 2007.

[20] N. Medvidovic and R. N. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering*, 26(1):70–93, 2000.

[21] M. Ouimet, K. Lundqvist, and M. Nolin. The timed abstract state machine language: An executable specification language for reactive real-time systems. In *Proceedings of the 15th International Conference on Real-Time and Network Systems*, 2007.

[22] S. Sentilles, A. Vulgarakis, T. Bures, J. Carlson, and I. Crnkovic. A component model for control-intensive distributed embedded systems. In *Proceedings of the 11th International Symposium on Component Based Software Engineering (CBSE2008)*, pages 310–317. Springer, Oct 2008.

[23] A. Vulgarakis and C. Seceleanu. Embedded systems resources: Views on modeling and analysis. In *COMPSAC*, pages 1321–1328, 2008.