REMES: A Resource Model for Embedded Systems

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

Abstract

In this paper, we introduce the model REMES *for formal modeling and analysis of embedded resources such as storage, power, communication, and computation. The model is annotated with both discrete and continuous resources. It is in fact a state-machine based behavioral language with support for hierarchal modeling, continuous time, and a notion of explicit entry and exit points, making it suitable for component-based modeling.*

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 to provide a basis for formal 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, 11, 12, 13, 17, 18, 19, 20]. 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, monotonic type (like energy), of non-monotonic type (like memory), or of discrete nature (e.g. I/O ports). A representative analysis goal is to answer the *feasibility* question: does the composition of the worst-case resource requirements of components stay within the available resources provided by the implementation platform?

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 we assume a repository of models, the designer might need, at some point, a refined component model, with modified behavior or more efficiently implementable data structures.

Assume the following scenario: suppose we start building up an embedded system for which we identify the interconnected software components as being $C_1, \ldots, C_n$ (see Figure 1). The dotted lines represent connections to other possible system components. We also assume that the hardware abstraction provides us with global available resources R. Consider that the computed resource requirement of C_1 is R_{C1}, of C_2, R_{C2} and so on. In Figure 1, the components are annotated with this information.

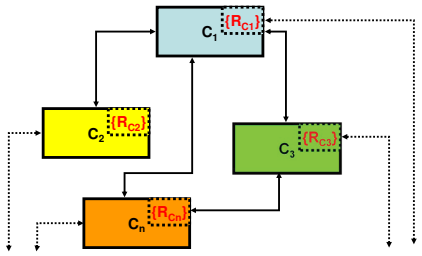


Figure 1. n-component Embedded System with resource annotations.

Suppose now that a different designer wants to use some component B, from the repository, instead of C_1 (for one of the reasons mentioned previously). So, we replace C_1 by B, both functionally and resource-wise. However, it so happens that B needs more resources than C_1 to perform its function: $R_B > R_{C1}$. Intuitively, the resource feasibility test will fail for the new composition, thus preventing us from accommodating B. In order to be able to include B in the system, we need to "fine-tune", in the sense of decreasing enough, the resource requirements of one or more components, for instance, by code-optimization. Then, by rechecking resource feasibility, we should get a positive an-

swer.

A more challenging situation arises if we do not have access to the components implementation. How can we then accommodate B? One could think of trying to change the communication between components, or maybe the allocation of components to hardware units. We would be interested to assess, before deployment, how would any of these design decisions affect the system overall resource consumption. This amounts to finding an appropriate trade-off between different configuration requirements and constraints.

Performing these kinds of analysis of the embedded system's resource usage, starting at an early stage of design, and up to an as close to implementation stage as possible, 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 some repository, choosing among various admissible architectural designs, or transforming a component model into one with less resource requirements.

In this paper, we propose a modeling framework and associated analysis techniques for performing quantitative analysis such as resource utilization/feasibility, trade-off, and optimal resource-utilization analysis. The model, called REMES, is tailored for embedded systems, but it is also 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, power, CPU, and buses. As such, the characterized and classified abstract resource types are not tied to any particular formal semantical interpretation.

In addition, REMES supports modeling of functionality and timing in a dense time state-based hierarchical modeling language, and we show how the modeling language and a number of important resource analysis problems can be analyzed in the framework of multi-priced timed automata. As such, REMES narrows the gap between architectural modeling and semantical analysis models. This claim is demonstrated in a case study, in which a temperature control systems is modeled and analyzed.

Related Work. In [21], we have presented related work on modeling and analyzing resources in embedded real time systems, which falls into three categories. First, research has been devoted to code-level resource modeling and analyzing, in component assemblies. In Koala [12] and Robocop [11] component frameworks, static memory estimation has been performed for applications in which the instantiated components in a composition are known before runtime. In real-world applications, the set of components may dynamically change, so the estimation will not be necessarily correct. Moreover, low-level code-driven resource estimates can only be used in cases when one has an access to the component's implementation. More abstract description on expected resource usage may be needed for not-yet implemented components or for guiding the selection of components from the repository. Second, some UML-based attempts [13, 6] to tackle the analysis of embedded resources have been carried out. Although graphical and intuitive, these approaches lack a formal description that could provide the designer with verified resource usage claims. Third, higher-level formal approaches [17, 18, 19] 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. Another formal approach, this time based on timed abstract state machines [20] describe resources as simple annotations, in the form of real-valued variable assignments, hence the framework can not support trade-off analysis of conflicting resource requirements.

The rest of this paper is organized as follows: in Section 2 we present preliminaries. In Section 3 we present a taxonomy of resource types, and introduce the REMES model for modeling resources in embedded systems. Its associated resource analysis problems are introduced and formalized in Section 4. In Section 5 we illustrate our approach on a temperature control system. Finally, we conclude the paper and present a line of future work, in section 6.

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,

$$(l, u) \stackrel{d,p}{\rightarrow} (l, u \oplus d)$$

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

$$(l, u) \stackrel{a,p}{\rightarrow} (l', u')$$

correspond to taking an edge $l \stackrel{g,a,r}{\rightarrow} l'$ 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

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

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 $a!$).

In order to specify properties of PTA, the logic Weighted CTL (WCTL) has been introduced [10]. WCTL extends Timed CTL with resets and testing of cost variables. We refer the reader to [10] for a thorough introduction 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 [10, 16]. 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) reaching a goal state. Note that the solution is a set of pairs, rather than a single pair, since the costs contributed from the individual costs can be incomparable, i.e., if for two traces (p_1, p_2) and (p'_1, p'_2) e.g., $p'_1 < p_1$ and $p_2 < p'_2$. In this setting, the minimal cost reachability problem is to find the set of pairs with minimum cost reaching the goal state. Dually, the maximization problem is defined as finding the set of pairs with 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 [16] for a thorough description of optimization problems in MPTA.

3 REMES: Our Resource Model for Embedded Systems

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 the resource usage that occurs instantaneous in time, whereas the *utilization* of a resource is the rate of consumption over time, i.e., the first derivative of consumption denoted $\dot{c}$. Both consumption and utilization can be of *monotonic* or *non-monotonic* nature, and resource consumption can further be considered as *discrete* or *continuous*. 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 B or C is continuous, which is in opposition to the discrete resource consumption nature for the resources from class A. The consumption for the resources from class A (memory, ports) and the utilization for the resources that make class C (CPU, bus bandwidth) is a non-monotonically increasing function, meaning that, e.g., memory or CPU may be released. In contrary, the resource power of class B, can not be recovered or released, hence its consumption is monotonically increasing. 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 resource consumption and utilization information, as well as other constructs that facilitate

Resource Class	Characteristics
A (memory, ports)	discrete $\dot{c} = 0$ consumption non-monotonically increasing referable
B (power)	continuous $\dot{c} = n$, $n \in \mathbb{N} \setminus \{0\}$ consumption monotonically increasing non-referable
C (CPU, bandwidth)	continuous $\dot{c} = n$, $n \in \mathbb{N} \setminus \{0\}$ utilization non-monotonically increasing non-referable

Table 1. Resource classes/characteristics

the application of REMES to modeling both functional and extra-functional behavior of (real-time) component-based systems.

In REMES, the 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 2). 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 2, that the entry and exit points are drawn as blank and filled circles, respectively. The variables of mode M are partitioned into *local* variables, (L_M), and *global* variables (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.

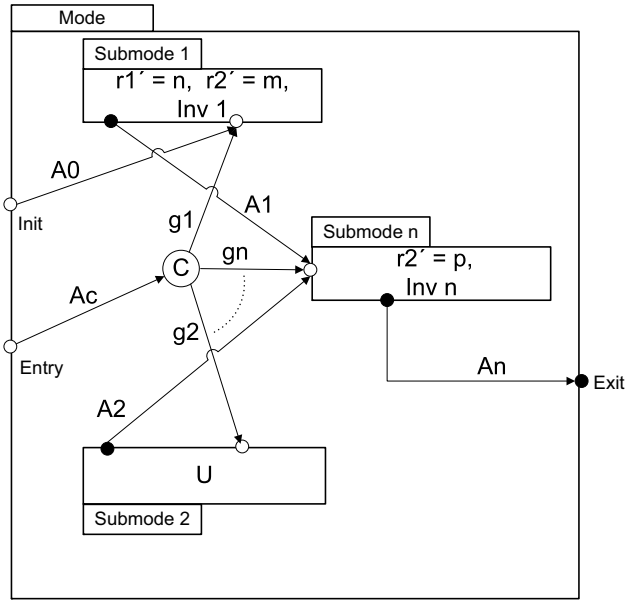


Figure 2. A REMES Composite Mode.

The atomic modes Submode 1 and Submode n in Figure 2 are annotated with their respective resource-wise continuous behavior, assuming that the corresponding component is utilizing resources $(r1, r2)$ belonging to class B or C. Such utilization is expressed by the first derivatives of the typed resource variables $r1 : T_B, r2 : T_C$, respectively, which give the rates at which the composite mode consumes the resources in time, depending on the executing submode.

For a composite mode the execution consists of a sequence of *actions* that connect its control points to the control points of its submodes. For example, in Figure 2, the parent mode fires the action labeled A_0, in order to execute Submode 1 after initialization, and similarly, actions $A_1, A_2, \ldots, A_n$, to further execute Submode 1, Submode 2, ..., Submode n, respectively.

REMES supports two types of actions, a *delay/timed* action and a *discrete* action. A delay action describes the continuous behavior of the mode, and its execution does not change the mode; on the other hand, executing a discrete action results in a mode change, via the exit point. Discrete actions are instantaneous actions. Observe that Submode 2 is decorated with letter *U*, meaning that such a mode exits right-away after its activation, without any delay. Such modes are called *urgent*.

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 action has been fired. 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 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 discrete actions.

Similar to Statecharts [14], REMES provides a *conditional connector* (depicted by C in Figure 2), which allows the selection of one discrete action from two or more possible ones, via the guarding boolean conditions (guards $g_1, g_2, \ldots, g_n$) on the discrete actions 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 taken and the component remains in its current mode, executing delay actions. If more than one guard are TRUE then one of the enabled discrete actions could be taken nondeterministically.

We classify a mode's discrete actions as follows:

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

Note that in REMES, as opposed to CHARON, each mode describes the behavior of an embedded component, and a composite mode is a way of encapsulating behavior and it describes a composite component. As such, there is no need to flatten the hierarchy.

Formal Definition of a Mode. A mode $M = (SM, V, In, Out, A, RC, Inv)$ is a tuple where:

- SM : the set of submodes,
- V : the set of variables,
- In : the set of entry control points,
- Out : the set of exit control points,
- A : the set of actions,
- RC: the set of resource constraints that define the admissible values for the utilization (first derivative of consumption) of the involved resources in class B or C,
- Inv : the set of invariants.

For the submodes of M, the following condition should hold: $G_{SM} \subseteq L_M \cup G_M$, 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 taking 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 submode is either connected to the exit point of the top-level mode, or deterministically connected to an entry point of another mode that eventually leads to the top-level mode's exit.

For example, in Figure 2, 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 exit 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 sub-modes can never execute in parallel; they simply obey the strict execution order imposed by the control flow.

We say that two REMES models, *A* and *B*, are *compatible* if $G_A = G_B$. In the following, we give the definition of mode composition.

Definition 2 *Assume A and B are two REMES components (modes). Then, the composition $D = A \,||\, B$ is the mode with the set of local variables $L_D = L_A \cup L_B$, the set of global variables $G_D = G_A = G_B$, and the set of top-level modes $Mode_D = Mode_A \cup Mode_B$.* ■

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 components 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 $r_1, r_2, \ldots, r_n$ denote the consumption of $R_1, R_2, \ldots, R_n$, 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, if we are designing a heavily resource-constrained soft real-time ES that might tolerate lateness at the expense of quality of service, and are considering trade-offs between memory consumption and (execution) time, we can assign higher weight to memory than to time.

Translating REMES into PTA. In order to be able to analyze REMES compositions, formally, we need a semantical translation of the model. If we consider resource consumptions $r_1, r_2, \ldots, r_n$ as cost variables $c_1, c_2, \ldots, c_n$, we can use the framework of Priced Timed Automata as the underlying semantical representation.

Translating REMES into PTA is quite straightforward: the syntactic REMES element of a discrete action corresponds to an edge in PTA, whereas the REMES semantical 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. In the following, we formalize some of the main analysis goals that we are interested in.

4.2 Feasibility Analysis

Component-wise feasibility analysis with respect to resource consumption requires (symbolic) algorithms on PTA, which compute the cost of the most "expensive" trace that will eventually reach some goal. The respective cost could then be compared against the allocated resources, per component, on the target platform. For resources like non-referrable memory and power, the composition of individual resource consumptions of REMES components is additive.

If we consider the PTA model of Definition 1, as our semantical translation of a REMES model, feasibility goals can then be formalized as the following WCTL properties:

$$E\,F_{cost \geq n}\, v \tag{1}$$

$$A\,G\,(q \Rightarrow E\,F_{cost \geq n}\, v) \tag{2}$$

where A, and E are the usual CTL universal and existential path quantifiers, respectively, and G and F are the CTL temporal operators "always" and "eventually", respectively.

The properties (1), (2) are in fact reachability and liveness properties, respectively, indexed with cost constraints; the first property states that there exists a path such that, eventually, some target location v is reached with a cost greater or equal then some given value n; the second property states that for all paths it is always the case that once a location is reached, there exists a way by which v will be eventually reached with a cost greater or equal than n. However, model-checking WCTL formulae is decidable just for one-clock priced automata with a stopwatch cost (cost with rates in $\{0,1\}$) [9]. For other PTA, one can only verify reachability properties of the form given by (1).

Such reachability goals require algorithms for synthesizing maximal reachability costs for PTA, and they have been proposed by Larsen and Rasmussen [16]. In the cost function $cost = w_1 \times c_1 + \ldots + w_n \times c_n$, $c_1, \ldots, c_n$ are constants, so the maximization problem reduces to maximizing a single cost variable representing the accumulated resource consumption of all resources of interest, regardless of the class they belong to. Hence, semantically, the various resources become undistinguishable.

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

4.3 Optimal Resource Utilization

The optimal resource utilization problem reduces to minimizing the one-cost function $cost = w_1 \times c_1 + \ldots + w_n \times c_n$, such that one of the following WCTL properties is satisfied:

$$E\,F_{cost \leq n}\, v \tag{3}$$

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

The only difference in properties (3), (4) from properties (1), (2) is the fact that the cost is bounded from above with a value that could represent the available resources. Similar to the feasibility case, only cost-optimal reachability properties can be verified by a model-checker. Later, we show how can the minimum cost-trace 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, can not be applied as such anymore, since minimal/maximal reachability analysis requires a monotonically increasing cost function. The optimal-cost reachability problem has been theoretically solved even when negative costs are involved [8].

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 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 B or C, the function $cost = w_1 \times c_1 + \ldots + w_n \times c_n$ becomes a multi-cost function that lets us distinguish between various types of resources (e.g., between power and CPU). This forces us to carry out the analysis on MPTA, rather than on PTA.

Assuming power and CPU as the resources of interest, we want to determine which are the simultaneously achievable pairs of costs (c_{pow}, c_{cpu}) such that power consumption is minimized, while CPU consumption remains bounded from above, or power consumption is maximized while CPU consumption is bounded from below. In WCTL, the properties to be satisfied would then be as follows:

$$E\,F_{c_{pow} \leq n}\, v \quad and \quad c_{cpu} \leq m$$

or,

$$E\,F_{c_{pow} \geq n}\, v \quad and \quad c_{cpu} \geq m$$

Such trade-off analysis can be carried out through conditional reachability verification on MPTA [15], by considering c_{pow} as the primary cost and c_{cpu} as the secondary cost. Larsen and Rasmussen have proved that such problems are decidable for MPTA [15].

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, depicted in Figure 3. 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.

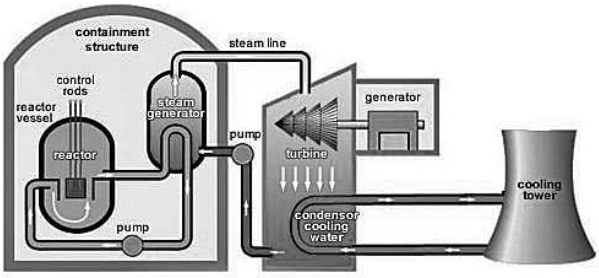


Figure 3. A heat producing reactor.

5.1 A REMES Model of TCS

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 4. 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.

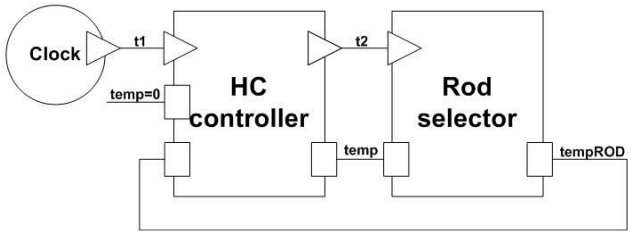


Figure 4. Component based TCS model.

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

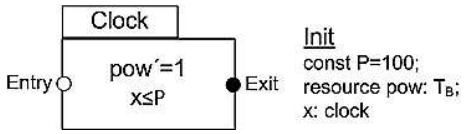


Figure 5. The Clock modeled in REMES.

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.

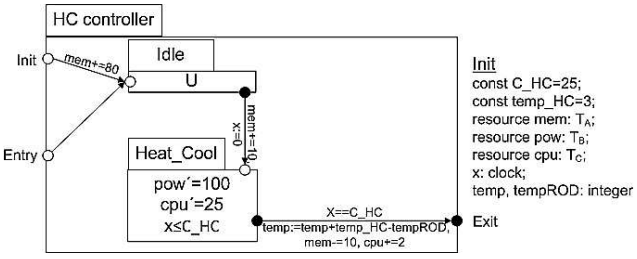


Figure 6. The HC Controller modeled in REMES.

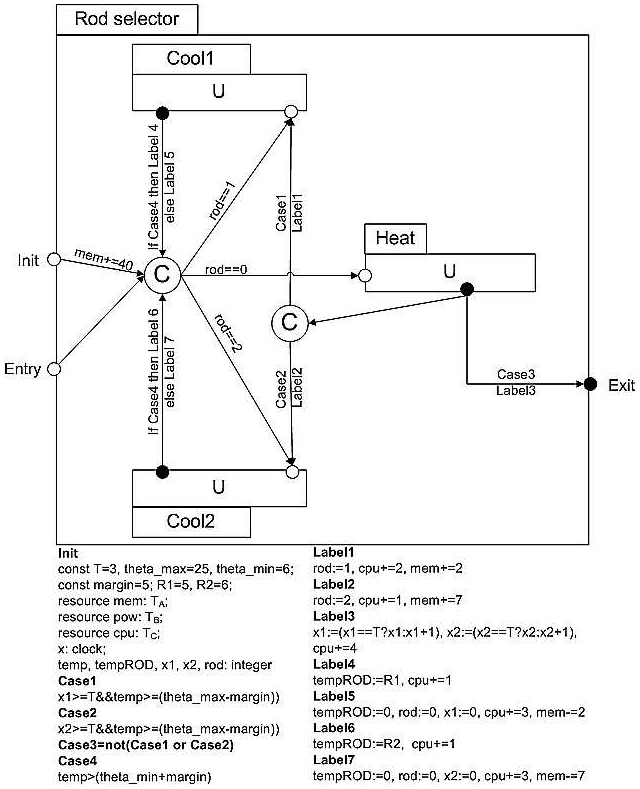


Figure 7. The Rod selector modeled in REMES.

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 5, 6, and 7, 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 up of submodes, conditional connectors, and discrete actions, as described in Section 3.

In the TCS model, we make use of three resources: memory, power, and CPU, which belong to the three different classes of the taxonomy presented in Section 3.1. We assume that every cpu instruction utilizes one cpu unit. 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.

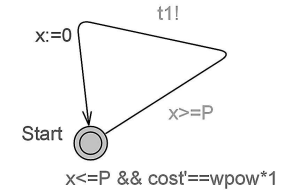
5.2 A PTA model of TCS

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 8. The declared variables and their initial values are shown in Table 2.

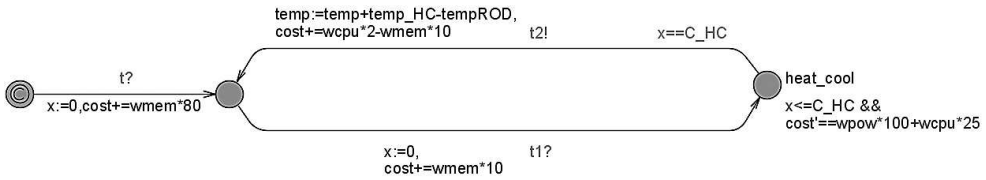
The Clock is modeled as a simple PTA that, after every P time units, periodically synchronizes on channel t1 with HC controller. The HC controller PTA has three locations: Start, Idle, and Heat_Cool. The constant C_HC is the execution time of the HC controller. The difference (temp_HC − tempROD), where 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

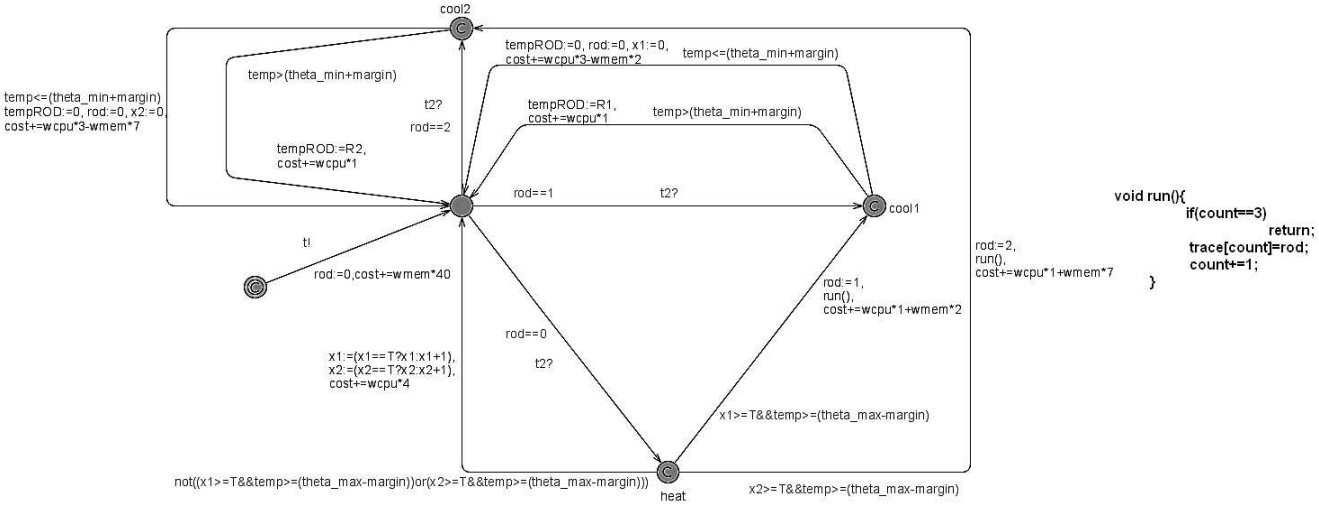
[1]See the web page www.uppaal.org/cora for more information about the UPPAAL CORA tool.



(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 8. TCS modeled with three PTAs.

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 8(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 power, CPU, and memory. We consider CPU to be the most critical resource, followed by memory. Power is not as critical, yet it is taken into consideration in order to impose higher energy efficiency in the system. Therefore, we give highest weight to CPU and lowest to power. 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

$$c_{tot} = \mathsf{wcpu} \times c_{cpu} + \mathsf{wmem} \times c_{mem} + \mathsf{wpow} \times c_{pow}$$

where wpow = 1, wcpu = 15, and wmem = 2, and c_{cpu}, c_{mem}, and c_{pow} are the consumed amounts of cpu, memory, and power, 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. 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 3 shows the cost of this best trace, and also

Scope	Declarations

Global	const int wpow = 1, wcpu = 15;
	const int wmem = 2;
	int temp = 7, tempROD = 0;
	chan t1, t2, t;

Clock	const int P = 100; clock x;
HC controller	const int C_HC = 25, temp_HC=3 clock x;

Rod selector	const int T = 3, theta_max = 25;
	const int theta_min = 6;
	const int margin = 5, R1 = 5, R2 = 6;
	int rod, count = 0, trace[3];
	int x1 = 3, x2 = 3;
	clock x;

Table 2. Declarations of the TCS PTA model.

the cost of another more expensive trace where only rod 2 has been used.

Scenario	Order of execution	Cost
1	P_2-P_2-P_1	253229
2	P_2-P_2-P_2	253239

Table 3. Cost of execution for different rods insertion scenarios.

For TCS, we can only partially tackle the trade-off resource analysis problem, by giving higher weight to the most critical resource, the CPU, followed by memory and power. 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.

6 Conclusions and Future Work

In this paper, we have introduced REMES — a formal 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 released, and whether they can be referred to, or not. Resources that can be naturally modeled include memory, ports, power, CPU, and busses etc.

In order to model usage of resources in a system, REMES has a behavioral modeling language influenced by CHARON [4], and is further borrowing ideas from timed and hybrid automata, and Statecharts [14]. The language supports hierarchical modeling and has a notion of explicit entry and exit points, making it suitable as a semantical basis in component based development frameworks. REMES has a notion of continuous variables, flows, and progress constraints (invariants), which makes it a suitable modeling language for timed behaviors in embedded systems.

In this setting, we have defined three important resource analysis problems: feasibility analysis, trade-off analysis, and optimal resource analysis. These problems are all defined over 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, reliability, or costs of the available resources.

To support 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 power, CPU, and memory resources. The system is architecturally modeled in the component modeling language SaveCCM, and REMES is used to model function, timing, and resource usage of the included components. To analyze the optimal resource usage of the system, we model the system and the weighted sum of resource costs, as a network of priced timed automata, and perform the analysis in the UPPAAL CORA tool.

As future work, we plan to solve the feasibility analysis problem for systems in which the global cost function is non-monotonically increasing. 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. We also plan to integrate REMES and its notion of resources in the recently suggested ProCom component model and its associated tools.

Acknowledgments: This work was partially supported by the Swedish Foundation for Strategic Research via the strategic research centre PROGRESS.

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] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

[14] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 91(1):231–274, 2003.

[15] 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.

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

[17] I. Lee, J.-Y. Choi, H.-H. Kwak, A. Philippou, and O. Sokolsky. A family of resource-bound real-time process algebras. In *FORTE*, pages 443–458, 2001.

[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] 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.

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