Using Improved Resource Interfaces to Formally Describe Adaptability in Embedded Systems

Magnus Persson and Martin Törngren
KTH (The Royal Institute of Technology), Stockholm, Sweden
magnus.persson@md.kth.se, martin@md.kth.se

ABSTRACT

In embedded systems, timing and resource utilization are vital aspects, having impacts on the deployed software. In this paper, an extensible formal framework for modeling of resource usage and adaptability of software components is introduced. The formalism explicitly supports modeling resources of various types and in different ways; and the timing requirements that applications have. The models can be used as a basis for quantitative analysis, and in the extension as a basis for system synthesis. An example of usage of the formalism is given.

Keywords

resource interfaces, resource modeling, adaptivity, formal methods, middleware, QoS, embedded real-time systems, timed automata, DySCAS, DyLite

1. INTRODUCTION

Real-time requirements often apply to software in embedded systems. At the same time, as typical embedded systems are put under considerable cost pressure, the hardware and software resources available are limited. These two facts together make well-working resource management necessary to satisfy performance requirements, either during design-time (e.g. off-line or fixed priority scheduling) or using an adaptive scheme (e.g. run-time negotiation for quality of service). Additional dynamics such as changing environments, varying resource availability and user configuration promote the latter [2, 7].

To provide guarantees of performance in these systems, there is a need for formal descriptions of resource requirements in adaptable computer systems. However, sufficiently powerful formalisms with this purpose do not yet exist:

> "The existing QoS contract languages are not equipped with a formal meaning, thus do not provide a basis for formal proofs, nor can they be used to perform symbolic computations." [7, section 13.6]

A first step towards such reasoning is to supply formalisms, which can be used to formulate problems and express requirements such as the design-time configuration problem or online adaptivity decisions.

As several applications normally share the same system, these descriptions also have to be *composable*, i.e. it should be possible to combine two specifications of separate components into a (sensible) specification of the compound component consisting of them.

1.1 Related Work

Most other approaches only put emphasis on one of these aspects, performing modeling either of timing *or* on resource usage. This brief survey focuses mainly on approaches aiming to model both timing issues *and* resources. More related work can be found in [18].

The SPEEDS project has defined the HRC component model [15], based on hybrid automata and providing several different possible views of components. Additionally, a good theoretical basis for contract composition is provided [4].

MARTE [17] is a UML profile aiming to provide modeling of embedded systems for UML. MARTE provides constructs for generalized modeling of non-functional properties, including timing and resource usage. The time modeling provided is based on a solid formalism, but for resources, the profile relies on external tools to define semantics.

Lusceta [6] is a toolsuite using timed automata to model QoS management policies, but is less formal for resources.

Hierarchical scheduling, as in e.g. [16], is an interesting parallel and possible source for concrete resource models.

2. A PROPOSED FORMALISM

In this paper, we propose a formal framework, building on the idea of *resource interfaces* [8] (an extension of timed interfaces [10]). We will however use a semantically different definition in this paper, additionally providing a way to also model soft deadlines, and use a more generalized way of modeling the resources. The intention is to make the formalism flexible, and hence it can be instantiated with different resource models, thus the term *formal framework*. The formalism is further described in an extensive report [18].

A timed automaton [1,3] is used to represent the behavior of each software component in the embedded system. Timing requirements are expressed as guard conditions in the timed automata. One or several software components correspond to a single task at the operating system or middleware level, hence also defining a task model for the platform. Each state represents a mode of operation of the software component and is further annotated with resource metrics based on special resource models.

2.1 Definition of Timed Automata

The below definition is based on established descriptions of timed automata, as described in e.g. [1, 3]. As the concept has evolved slightly over time, the formal definition used below is a synthesis building on both referred papers.

A timed automaton $\mathcal{A}$ is a tuple $\langle \mathbf{\Sigma}, \mathbf{S}, s_0, \mathbf{C}, \mathbf{G}, \mathbf{E}, \mathbf{I} \rangle$, representing a *timed language*[1] $\mathcal{L}_{\mathcal{A}}$, where

- $\mathbf{\Sigma}$ is a finite alphabet,
- $\mathbf{S}$ is a finite set of states,
- $s_0 \in \mathbf{S}$ is the starting state
- $\mathbf{C}$ is a finite set of clock variables,
- $\mathbf{G}$ is a finite set of *guard variables* of one or several well-defined types (e.g. boolean, integers),
- $\mathbf{E} \subseteq \mathbf{S} \times \mathbf{S} \times \mathbf{\Sigma} \times \mathbf{C} \times \mathbf{G} \times \mathbf{\Phi}(\mathbf{C}, \mathbf{G})$ gives the set of transitions. An edge $\langle s, s', \sigma, \mathbf{c}, \mathbf{g}, \phi \rangle$ is used to represent a transition from state s to s' on input signal (word) $\sigma \in \mathbf{\Sigma}$. The clocks to be reset are given by the set $\mathbf{c} \subseteq \mathbf{C}$, guard variables that will receive updated values are given by the set $\mathbf{g} \subseteq G$, and $\phi \in \mathbf{\Phi}(\mathbf{C}, \mathbf{G})$ is the guard expression for the transition,
- $\mathbf{I} : S \rightarrow \Phi(\mathbf{C})$ assigns invariants to states. Invariants are guard expressions, but applied to states instead of transitions.

A guard expression is a logical expression evaluating to either *true* or *false*, built of clock variables, guard variables and constants. Guard expressions are evaluated *after* guard variables have been updated.

A common operation on automata in general is synchronous composition, $\mathcal{A}||\mathcal{B}$, informally defined as the largest possible automaton representing the constraints on $\mathcal{L}_{\mathcal{A}}$ and $\mathcal{L}_{\mathcal{B}}$ represented by $\mathcal{A}$ and $\mathcal{B}$, taking commonalities in their alphabets into consideration. Due to space constraints, we refer to the previously mentioned references for a formal definition.

2.2 Resource Models

In addition to normal timed automata, we annotate each state of the timed automata with resource metrics, based on one or several mathematically founded *resource models* $\mathbf{R}_1, \mathbf{R}_2, \ldots$. This means, there is a one-to-one mapping between the states s_i to vectors with resource metrics, $\mathbf{r}_{s_i} = \langle r_{1,s_i}, r_{2,s_i}, \ldots \rangle$. The resource models need to be supplied by the user of the modeling framework (or by a standard library of resource models). As each resource model is supposed to denote values that are to be compared with each other, each resource resource model $\mathbf{R}_i$ forms a partially ordered set. For two resource metrics, the one that applies the strictest requirements is considered smaller.

[1]A timed language is defined as a sequence of words, ordered by time, each paired with the time at which it occurs.

Further, for each of these resources, two functions need to be defined: $\oplus$ and $\otimes$. Informally, $r_A \oplus r_B$ denotes a resource metric that represents an amount of resources that is always lower or equal to the amount of resources represented by *either* r_A or r_B, while $r_A \otimes r_B$ represents the amount of resources necessary to accommodate *both* r_A and r_B. It is also assumed that both these operations are commutative. Formally, these requirements can be described like follows:

$$\oplus : \mathbf{R}^2 \rightarrow \mathbf{R} \qquad \otimes : \mathbf{R}^2 \rightarrow \mathbf{R}$$

$$r_A \oplus r_B \leq r_A \quad r_A \oplus r_B \leq r_B \quad r_A \otimes r_B \geq r_A \quad r_A \otimes r_B \geq r_B$$

$$r_A \oplus r_B = r_B \oplus r_A \qquad r_A \otimes r_B = r_B \otimes r_A$$

If several terms/factors are needed for the operations, we use the alternative notation $\sum r$ and $\prod r$.

The conventional definition of synchronous composition of timed automata is also extended to allow for automata that are resource-annotated: for a composed state $s_{\mathcal{A}||\mathcal{B},ij}$, the resulting resource annotation to use is $r_{\mathcal{A},i} \otimes r_{\mathcal{B},j}$.

2.3 Time-Dependent Utility Functions

A common way of formalising soft real-time requirements is to use utility functions instead of hard deadlines. Based on the idea described by Jensen [14], the two concepts can be applied jointly, where utility functions over time $U(t)$ denote the utility value of a computation, and utility values equal to $-\infty$ denote hard timing requirements. To simplify implementation together with timed automata, we choose to instead define the utility functions over the clock variables: $U(c_1, c_2, \ldots c_n)$.

These utility functions can be used to choose between alternative configurations, using some sort of evaluation criteria, e.g. absolute or weighted sums, averages, or other mathematical functions.

2.4 Mapping to Architectural Description

We have now defined the modeling abstractions that we will need, and will now give a mapping to concepts used in traditional software engineering. A software component is represented by a single annotated automaton. Timing constraints are expressed as clock constraints that apply to the transitions of the automaton. Tasks typically consist of one or several components, and are hence represented by synchronously composed automata. These automata share clocks in case timing constraints are applicable to groups of tasks (e.g. timing chains). The synchronous composition of all timed automata represent the entire software system.

Resources (both hardware and software), such as CPUs, networks and semaphores, are each represented by a single resource model and a resource limit. A networked system would for example be modeled with one resource for each CPU and one for the network. Arbitration protocols for resources may, depending of the level of modeling abstraction, either be an implicit part of the resource models (which then have to explicitly take account for any overhead caused by the arbitration protocol), or be explicitly modeled by a timed automaton.

Adaptability can be modeled in two ways, depending on when it occurs. Run-time adaptability has to always be modeled as possibilities to execute the automaton differently; e.g. giving the execution platform a choice between two alternative transitions, or by making it possible to delay a certain transition. These different execution patterns cause different resource utilization and may be used to represent e.g. different execution frequencies or migrating a software component between different processors. Design-time adaptability can alternatively be modeled as the alternative use of different timed automata, either alternative representations for the same software component or of two alternative components.

2.5 Formalizing the Configuration Problem

We now can use the above description of the system and the software components, to describe the configuration problem, i.e. the problem of finding a valid system configuration. This can conceptually be viewed as iterating over configurations until a feasible one is found, i.e. one where the applications' resource demands are met by available resources:

$$\forall r_i, s \in \mathcal{A}||\mathcal{B}||\mathcal{C}\dots \text{ it holds that } r_{i,s} \leq r_{i,\max}$$

where i is the resource index, $r_{i,\max}$ is the maximum usage limit for each resource, and, implicitly, the resulting composed timed automaton $\mathcal{A}||\mathcal{B}||\mathcal{C}\dots$ is well-defined, in the sense that it does not represent a deadlock situation.

The above criterion gives a feasibility criterion; given two feasible configurations, it is also relevant to be able to compare them to each other to find out which is preferable over the other. To choose between them, some sort of optimization criteria, based on the utility functions described above, needs to be applied.

3. AN EXAMPLE INSTANTIATION

As an example on how the framework can be applied, we consider the DyLite middleware [19, 20], a compact middleware framework supplying simple resource management for reconfigurable applications on small microcontrollers, developed during the DySCAS[2] project [11].

3.1 Resource Models

We describe two resource models used in DyLite[3]:

- CPU time, modeled using real-valued utilization values and Liu-Layland analysis[4], and
- memory usage, modeled as integers.

We also need to give the exact definition of the operations $\oplus$ and $\otimes$ for the two resource models. For both, we use traditional addition for $\otimes$ and the minimum operation for $\oplus$:

$$r_{\text{CPU}} \in \mathbf{R}_{\text{CPU}} \mapsto \mathbb{R}_+ \qquad r_{\text{mem}} \in \mathbf{R}_{\text{mem}} \mapsto \mathbb{N}_0$$

$$r_{\text{CPU},A} \oplus r_{\text{CPU},B} \triangleq \min r_{\text{CPU},A}, r_{\text{CPU},B}$$

[2]DySCAS means *Dynamically Self-Configurable Automotive Systems*.
[3]DyLite also defines a third: network bandwidth, which is not covered here.
[4]All applications are assumed to have appropriate priorities assigned.

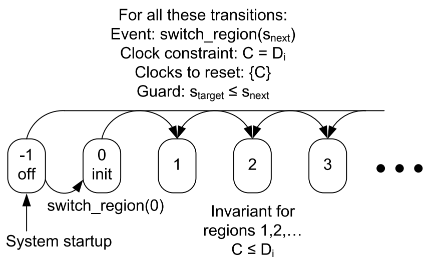
For all these transitions:
Event: switch_region(s_{next})
Clock constraint: C = D_i
Clocks to reset: {C}
Guard: $s_{target} \leq s_{next}$
-1
off
0
init
1
2
3
• • •
switch_region(0)
System startup
Invariant for
regions 1,2,...
C ≤ D_i

Figure 1: A timed automaton representing the DyLite task model.

QoS mode	Resource Usage		Deadline	Utility
s_i	r_{CPU}	r_{mem}	D_i	$U(C)$
1	2,3%	300 B	200 ms	3
2	6,1%	720 B	400 ms	13
3	6,5%	832 B	100 ms	14

Table 1: Example of resource annotations in DyLite.

$$r_{\text{CPU},A} \otimes r_{\text{CPU},B} \triangleq r_{\text{CPU},A} + r_{\text{CPU},B}$$

$$r_{\text{mem},A} \oplus r_{\text{mem},B} \triangleq \min r_{\text{mem},A}, r_{\text{mem},B}$$

$$r_{\text{mem},A} \otimes r_{\text{mem},B} \triangleq r_{\text{mem},A} + r_{\text{mem},B}$$

In the actual implementation, there are multiple instances of the CPU and memory resources (one for each connected node), but to simplify presentation, we will here limit ourselves to a single node.

3.2 Models of Components and Tasks

In DyLite, each component exactly corresponds to one operating system thread. For each task, at least one normal operating mode (QoS region) is defined. For each region, there is a deadline by which the the application should have called a special function (`switch_region(next_region)`) in the DyLite API, to signal that the code to be run within that region has finished. The modes are ordered; the assumption is that a higher ordered mode will always both provide a better user experience and use more of each resource.

A timed automaton representing this task model is depicted in Figure 1. The regions are $\{s_{-1}, s_0, s_1, \dots, s_n\}$. The set of clocks consists of a single clock, i.e. $\mathbf{C} = \{C\}$. For each outgoing transition, an exact deadline D_i for each mode s_i is supplied. Each region from s_1 and on is also annotated with resource usage $(r_{\text{CPU}}, r_{\text{mem}})$. The resource usages in s_0 and s_{-1} are assumed to be 0. An example of such annotations is given in Table 1. Finally each region is also annotated with a fixed utility value $U(C)$ for each state, which applies when $C < D_i$. Please note that all these restrictions only apply in the DyLite example and not to the framework in general.

3.3 Resource Constraints and Algorithm for Self-Configuration

For each resource, maximum resource levels are given. These are simply:

$$r_{\text{CPU},\max} \triangleq \lim_{n\to\infty} n(\sqrt[n]{2} - 1) \approx 0.69 \text{ (Liu-Layland border)}$$

$$r_{\text{mem},\max} \triangleq \text{ amount of memory on the node in question}$$

Using these borders, the reconfiguration algorithm [12, 13] implemented within DyLite is able to search for a feasible configuration, optimizing the sum of all utility values.

4. ALTERNATIVE INSTANTIATION WITH OTHER RESOURCE MODEL

There are many other possible resource models. Below, a couple of easily imaginable examples are given.

The first one is based on the hyperbolic bound [5], which gives the utilization limit as $\prod_{i=1}^{n}(U+1) \leq 2$, and provides an alternative resource model for processor utilization, with a more exact boundary:

$$r_{\text{CPU}} \in \mathbf{R}_{\text{CPU}} \mapsto \mathbb{R}_{+} \qquad r_{\text{CPU,max}} \triangleq 1$$

$$r_{\text{CPU},A} \otimes r_{\text{CPU},B} \triangleq (r_{\text{CPU},A}+1)\cdot(r_{\text{CPU},B}+1)-1$$

If DyLite had used easily exchangeable resource models, this alternative and more exact resource model could have transparently replaced the classical Liu-Layland analysis.

Other possible resource models can be defined over intervals ($[r_a, r_b]$, where $r_b > r_a$ and $r_a, r_b \in \mathbb{R}_{+}$), functions over time ($r(t)$, e.g. as in [9]), n-tuples ($\langle r_a, r_b, \ldots\rangle$) etc.

5. DISCUSSION

In this paper, a proposed formal framework has been introduced, able of describing both the real-time requirements that apply to a software component, and the resources that the task will consume. An example of how the formalism can be applied to an existing implementation of a resource-aware middleware has also been provided. Online adaptability can be modeled by using alternative transitions between states in the automata describing applications. Offline adaptability can additionally be represented by providing alternative automata to choose among during the system design.

5.1 Future Work

There are several lines of possible future work:

- Integrating support in tools for development of embedded systems, i.e. making it possible to directly in an architectural tool specify resources, tasks, timing constraints and modes, verify and analyse the system.
- Extending existing timed automata analysis tools (e.g. UPPAAL [21]) to support the formalism. This can probably be performed by generating additional automata for each resource, constricting any transition that would imply too high resource usage, using constraints solving.
- Evaluation in actual design work. What are the implications on design-time and run-time overheads to deploy the formalism? Is it a suitable abstraction for designers? Can modeling be done efficiently?
- How are models of the same system using different resource models related? How can translations between different framework instantiations be built?
- Further development of suitable algorithms for configuration optimization and optimization criteria, both for off-line design work and online decision evaluation.

6. REFERENCES

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

[2] R. J. Anthony, P. Ward, D. Chen, J. Hawthorne, M. Pelc, A. Rettberg, and M. Törngren. A middleware approach to dynamically configurable automotive embedded systems. In *Proc. of The First Annual International Symposium on Vehicular Computing Systems*, Dublin, Ireland, July 22 – 24 2008.

[3] G. Behrmann, A. David, and K. G. Larsen. A tutorial on Uppaal. In M. Bernardo and F. Corradini, editors, *Formal Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer, Communication and Software Systems (SFM-RT 2004)*, volume 3185 of *LNCS*, pages 200–236, Bertinora, Italy, Sept. 13–18 2004. Springer Verlag. an updated version of this paper is available from `http://www.it.uu.se/research/group/darts/papers/texts/new-tutorial.pdf`.

[4] A. Benveniste, B. Caillaud, A. Ferrari, L. Mangeruca, R. Passerone, and C. Sofronis. Multiple viewpoint contract-based specification and design. In *Formal Methods for Components and Objects*, volume 5382 of *LNCS*, pages 200–225, 2008. revised paper, presented at 6th International Symposium FMCO 2007, Amsterdam, Netherlands, Oct. 24–26.

[5] E. Bini, G. C. Buttazzo, and G. M. Buttazzo. Rate monotonic analysis: the hyperbolic bound. *IEEE Transactions on Computers*, 52:933–942, 2003.

[6] L. Blair, G. S. Blair, A. Andersen, and T. Jones. Formal support for dynamic QoS management in the development of open component-based distributed systems. *IEE Proc. Software*, 148(3):89–97, June 2001.

[7] B. Bouyssounouse and J. Sifakis, editors. *Embedded Systems Design: The ARTIST Roadmap for Research and Development.* Number 3436 in LNCS. Springer Verlag, 2005.

[8] A. Chakrabarti, L. de Alfaro, T. A. Heinzinger, and M. Stoelinga. Resource interfaces. In *Embedded Software (Proc. from Third International Conference EMSOFT)*, volume 2855 of *LNCS*, pages 117–133, Philadelphia, PA, USA, Oct. 13–15 2003.

[9] S. Chakraborty, Y. Liu, N. Stoimenov, L. Thiele, and E. Wandeler. Interface-based rate analysis of embedded systems. In *RTSS '06: Proc. of the 27th IEEE International Real-Time Systems Symposium*, pages 25–34, Washington, DC, USA, 2006. IEEE Computer Society.

[10] L. de Alfaro, T. A. Heinzinger, and M. Stoelinga. Timed interfaces. In *Proc. of Embedded Software (EMSOFT)*, pages 108–122, Grenoble, France, Oct. 7–9 2002.

[11] DySCAS Consortium. DySCAS project website. http://www.dyscas.org.

[12] L. Feng, D. Chen, M. Persson, T. Naseer Qureshi, and M. Törngren. Dynamic configuration and quality of service in automotive embedded systems. Technical Report TRITA MMK 2008:12, ISSN 1400-1179, ISRN/KTH/MMK/R-08/12-SE, Stockholm, 2008.

[13] L. Feng, D. Chen, and M. Törngren. Self configuration of dependent tasks for dynamically reconfigurable

automotive embedded systems. In *Proc. of 47th IEEE Conference on Decision and Control (CDC)*, pages 3737–3742, Cancún, Mexico, Dec. 9 – 11 2008.

[14] E. D. Jensen. Eliminating the 'hard'/'soft' real-time dichotomy. *Computing & Control Engineering Journal*, 8(1):15–19, Feb. 1997.

[15] B. Josko, Q. Ma, and A. Metzner. Designing embedded systems using heterogeneous rich components. In *Proc. of the INCOSE International Symposium*, Utrecht, Netherlands, June 2008.

[16] T. Nolte, M. Nolin, and H. Hansson. Hierarchical scheduling of CAN using server-based techniques. In L. Almeida, editor, *Proc. of the 3rd International Workshop on Real-Time Networks (RTN'04) in conjunction with the 16th Euromicro International Conference on Real-Time Systems (ECRTS'04)*, pages 27–30. Universidade de Aveiro, Campo Universitario de Santiago, 3810-193 Aveiro, Portugal, ISBN 972-789-136-5, June 2004.

[17] Object Management Group (OMG). A UML profile for MARTE: Modeling and analysis of real-time embedded systems.

[18] M. Persson. A timed automata formalism for modeling resource management and quality of service in real-time contexts. Technical report, Stockholm, Sweden, 2009.

[19] M. Persson, J. García, L. Feng, D. Chen, T. Naseer Qureshi, and M. Törngren. DyLite: Design, implementation and experiences. Technical Report TRITA MMK 2009:06, ISSN 1400-1179, ISRN/KTH/MMK/R-09/06-SE, Stockholm, Sweden, 2009.

[20] J. Söderberg, D. Scholle, J. Lövqvist, L. Krantz, M. Persson, J. García, C. Pihl, J. Granath, J. Tang, I. Drüke, V. Friesen, and M. Törngren. D3.3 DySCAS demonstrator application and specification. Dyscas project deliverable, 2008. project no. FP6-IST-2006-034904.

[21] UPPAAL. http://www.uppaal.com.