Integrating Behavioral Descriptions into a Component Model for Embedded Systems [1]

Aneta Vulgarakis, Séverine Sentilles, Jan Carlson, and Cristina Seceleanu
Mälardalen Real-Time Research Centre, Mälardalen University, Västerås, Sweden
{aneta.vulgarakis, severine.sentilles, jan.carlson, cristina.seceleanu}@mdh.se

Abstract

When component-based development is applied to distributed embedded systems, which are often safety-critical and subject to real-time constraints, it is of significant importance that reliable predictions of functional and extra-functional properties can be derived at design-time. Preferably, analysis should be performed in early development phases, where the cost of modifying the design is lower. Centered on an example application from the automation domain, we show how a component model specifically intended for embedded systems can be combined with a language for high-level formal behavior modeling. This permits analysis of system properties, while also supporting reuse of behavioral models when components are reused.

1 Introduction

An embedded system involves computation that is subject to physical constraints. Consequently, to ensure its predictable behavior, the embedded system design needs to be formally checked against different requirements due to various kinds of constraints including functional, timing, safety, and resource-driven (energy, memory etc.) constraints.

Designing an embedded system in a component-based manner, by building it from well-specified and verified components, intends to lower its complexity, in terms of implementation, but also modeling and analysis. In this paper, we describe the architecture of an embedded system by employing the real-time component model ProCom [14]. ProCom contains two layers, *ProSys*, and *ProSave*, each addressing the concerns of a different level of granularity.

In ProCom, the components, their services, ports, subcomponents, etc. can be annotated with various functional and extra-functional characteristics, represented as *attributes*. The attributes may be as simple as numbers (e.g., static memory usage of a component), but also as complex as intricate models. The ProCom's attribute framework [13] provides a systematic way of managing and integrating extra-functional properties, during the development of a component, or a system.

The possibly complex extra-functional behavior of ProCom components is modeled in a dense-time, state-based hierarchical language called REMES [12]. REMES is suited for modeling timing and resource-wise behaviors of embedded system components described by *modes*.

In this paper, we show how to pack a ProSys component, annotated with attributes such as required resources, with its associated REMES behavioral model. Then, both the interface and internal models of component behavior are seen as the actual reusable unit of composition, which can be employed as such, without modification, in adequate design contexts. To accomplish this, in Section 2, we propose a way of mapping the ProSys component interface, consisting of incoming and outgoing messages used for asynchronous communication, onto the entry and exit variables of REMES modes, respectively, such that the two models become connected.

To ensure predictable behavior, and guarantee behavioral correctness, we also show how to support formal analysis of various properties belonging to different categories (functional, safety, timing, resource-usage), within our favorite framework. This capability, together with the component-based design approach, might improve the efficiency of embedded system design, by allowing reuse of both models and analysis results. We exemplify the modeling and analysis approach on a *turntable* example system, modeled as a collection of ProSys components that we connect to their behavioral REMES models. The latter are then formally analyzed against functional, safety and resource-related properties, as described in Section 3, on the underlying Priced Timed Automata models.

An extended version of this paper is available as a technical report [16].

[1]This work was partially supported by the Swedish Foundation for Strategic Research via the strategic research centre PROGRESS and by the European Union under the ICT priority of the 7th Research Framework Programme in the context of the Q-ImPrESS research project.

2 Integrating REMES into ProCom

ProCom has been developed to facilitate the expression and analysis of functional and extra-functional properties, but does not, per se, provide any means to actually model them. It needs to be complemented with formalisms, complying with the component-based approach, that enable early formal analysis of relevant concerns. One step towards this support for formal analysis is the integration of REMES, by which functional behavior, resource consumption and timing can be addressed in a single modeling formalism.

Concretely, the integration is achieved by defining a new attribute type in the attribute registry. The attribute type can be attached to ProSys subsystem components, and has an attribute value consisting of a reference to the REMES model file in the component structure.

The relation between the ports of the component and the variables in the REMES model is given by a mapping, described below. As detailed in Section 2.2, we also need to slightly extend the REMES language to fit the active, non-terminating semantics of the ProSys components.

2.1 Connecting component interfaces and REMES modes

The connection between ProSave and REMES is done by mapping ProSave- to REMES interface. Each ProSave trigger port is mapped to a REMES boolean variable, and each ProSave data port is mapped to a REMES data variable of same type as the port type. In our previous work [12], we describe the connection between ProSave components and REMES modes on an abstracted version of a temperature control system. Here, we instead focus on connecting REMES to ProSys components.

Let P be the set of message ports of a ProSys component C. Each port $\mathsf{p}_{\mathsf{i}\in[1\ldots\mathsf{n}]} \in \mathsf{P}$ is a tuple $(\mathsf{Name}, \mathsf{Kind}, \mathsf{Type}, \mathsf{Value})$, where: Name is the port identifier, Kind models the input/output feature of the message port, Type encodes the port's data type, and Value stores the port's actual data value. Further, let M be a REMES mode that depicts the behavior of component C, and V the set of all variables of mode M that correspond to the ports of C. Each variable $\mathsf{v}_{\mathsf{j}\in[1\ldots\mathsf{n}]} \in \mathsf{V}$ is a tuple $(\mathsf{Name}, \mathsf{Kind}, \mathsf{Type}, \mathsf{Value})$, where: Name is an identifier of the variable, Kind distinguishes between read (global variable of the mode that may be written by other modes) and write (global variable of the mode that may be read by other modes) variables, Type encodes the variable's data type, and Value stores the actual variable value. The connection between mode M and the interface of component C is given by a mapping function $\mu : P \rightarrow V$ that maps component ports to REMES mode variables. Assuming non-empty messages ($\mathsf{Value}(\mathsf{p}_\mathsf{i}) \neq \mathsf{NULL}$), the mapping is defined as follows:

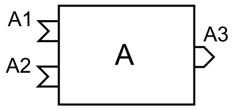
$$\mu(\mathsf{p}_\mathsf{i}) = \overline{\mathsf{v}}_\mathsf{i}, \quad \overline{\mathsf{v}}_\mathsf{i} = (\mathsf{v}_{\mathsf{i}_1}, \mathsf{v}_{\mathsf{i}_2}),$$

such that the following boolean condition holds:

$$\begin{aligned}
&\mathsf{Name}(\mathsf{v}_{\mathsf{i}_1}) = \mathsf{Name}(\mathsf{p}_\mathsf{i}) \wedge \mathsf{Name}(\mathsf{v}_{\mathsf{i}_2}) = \mathsf{Name}(\mathsf{p}_\mathsf{i}) + \text{``\_value''}\\
&\wedge\\
&((\mathsf{Kind}(\mathsf{p}_\mathsf{i}) = \mathsf{input} \wedge \mathsf{Kind}(\mathsf{v}_{\mathsf{i}_1}) = \mathsf{read} \wedge \mathsf{Type}(\mathsf{v}_{\mathsf{i}_1}) = \mathsf{bool}\\
&\quad \wedge \mathsf{Value}(\mathsf{v}_{\mathsf{i}_1}) = \mathsf{false} \wedge \mathsf{Kind}(\mathsf{v}_{\mathsf{i}_2}) = \mathsf{read}\\
&\quad \wedge \mathsf{Type}(\mathsf{v}_{\mathsf{i}_2}) = \mathsf{Type}(\mathsf{p}_\mathsf{i}) \wedge \mathsf{Value}(\mathsf{v}_{\mathsf{i}_2}) = \mathsf{Value}(\mathsf{p}_\mathsf{i}))\\
&\vee\\
&(\mathsf{Kind}(\mathsf{p}_\mathsf{i}) = \mathsf{output} \wedge \mathsf{Kind}(\mathsf{v}_{\mathsf{i}_1}) = \mathsf{write} \wedge \mathsf{Type}(\mathsf{v}_{\mathsf{i}_1}) = \mathsf{bool}\\
&\quad \wedge \mathsf{Value}(\mathsf{v}_{\mathsf{i}_1}) = \mathsf{false} \wedge \mathsf{Kind}(\mathsf{v}_{\mathsf{i}_2}) = \mathsf{write}\\
&\quad \wedge \mathsf{Type}(\mathsf{v}_{\mathsf{i}_2}) = \mathsf{Type}(\mathsf{p}_\mathsf{i}) \wedge \mathsf{Value}(\mathsf{v}_{\mathsf{i}_2}) = \mathsf{Value}(\mathsf{p}_\mathsf{i})))
\end{aligned}$$

In case an empty message is received/sent ($\mathsf{Value}(\mathsf{p}_\mathsf{i}) = \mathsf{NULL}$), the mapping function returns $\overline{\mathsf{v}}_\mathsf{i} = (\mathsf{v}_{\mathsf{i}_1}, \mathsf{NULL})$.

The parallel composition of the REMES modes associated to all ProSys components in the given system, together with representations of the ProSys message channels and connections, describe the whole system's behavior. Figure 1 exemplifies the mechanism of connecting the ProSys and REMES interfaces. Component A receives a message of type float via input port A1 and an empty message via input port A2, and it sends a message of integer type through output port A3.



ProSys port	REMES variables
A1	bool A1 and float A1_value
A2	bool A2
A3	bool A3 and int A3_value

Figure 1. Example of how ProSys ports are mapped to REMES variables.

2.2 REMES extensions

The traditional REMES modes described in [12] run to completion, and as such are suitable for depicting the behavior of ProSave components. In order to be able to capture the active behavior of a ProSys component, yet at the same time to ensure the termination of the internal behavior of a REMES mode, each REMES composite mode is enriched with a special write global variable called *history*, similar to CHARON [1]. Whenever an execution of a REMES composite mode would return to an already visited submode, the composite mode is exited and the control

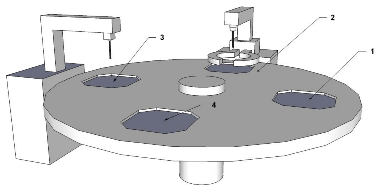


Figure 2. The turntable system (load and unload stations are not shown).

state of that mode is recorded into its history variable. Next time when the composite mode is entered, the control state of that mode is restored according to the value of the history variable. The history variable of a composite mode M contains the names of the submodes of M as values, or a special value null, which denotes that the mode is not active. A submode SM of a composite mode M is called active when the history variable of M has the value SM. Additionally, in the extended REMES for modeling behavior of ProSys systems, guards are modeled as conjunctions of boolean expressions over usual variables, to which constraints on history variables are added. Every time when there is a change in the history variable value of a certain composite mode its reactivation should be ensured. Access to the proper history variable values of every composite mode is done by System[name], where name denotes the name of the composite mode.

In addition, we have enriched REMES with a so called *non-lazy* mode. A non-lazy mode does not contain any invariant to specify how long it is allowed to delay in that mode. Time is allowed to pass in a non-lazy mode until at least one of the guards of the outgoing discrete actions evaluates to true. As such, in order to ensure the exit of a non-lazy mode, the disjunction of the action guards associated to the outgoing edges of that mode should always eventually become true.

3 Example

As an example, we have considered the turntable drilling system (from [4]). The system, depicted in Figure 2, consists of a rotating table that moves products between processing stations where they are drilled, tested and removed from the table once they pass the test. The table has four slots, each capable of holding one product. Thus, the stations can operate in parallel, so that while the first piece is being tested, the second piece can be drilled and a third piece loaded, etc.

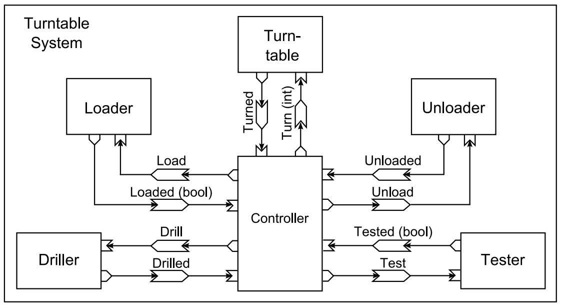


Figure 3. ProCom design of the turntable system.

3.1 Architecting the turntable in ProCom

We model the turntable drilling system in the ProCom component model, with five subsystems – *Loader*, *Driller*, *Tester*, *Unloader* and *Turntable* – as depicted in Figure 3. We assume that the *Loader* and *Unloader* components can be reused from a previous project. The *Controller* keeps track of the current status of the four slots, and activates the four stations and the turntable accordingly, by sending messages and receiving messages back once they are done.

It is possible to further decompose each of these ProSys components into either smaller ProSys components or into ProSave components according to the level of complexity of the functionality, and the potential for distribution. Before doing that, however, the developer may want to validate the feasibility of the design so far. Some properties can be analyzed from the ProCom design alone, for e.g., that connected ports and channels match, but in order to reason about for e.g., functional correctness, timing and resource consumption, we need to model the behavior of the components identified so far.

3.2 Behavior modeling in REMES

We model the functional, timing and resource usage behavior of the turntable components as models in REMES. Since the *Loader* and the *Unloader* components are reused, they already have behavioral models, but the remaining components should be given REMES models. Because of space limitation, here we only present the REMES model of the most complex component *Controller*, depicted in Figure 4. We refer the reader to [16] for details about behavioral modeling of the *Driller* component.

The *Controller* component keeps track of the states of the four slots and operates the stations and the turntable accordingly by exchanging messages with all of them. The behavior defined by the REMES mode consists of two main submodes, one in which the controller waits for messages

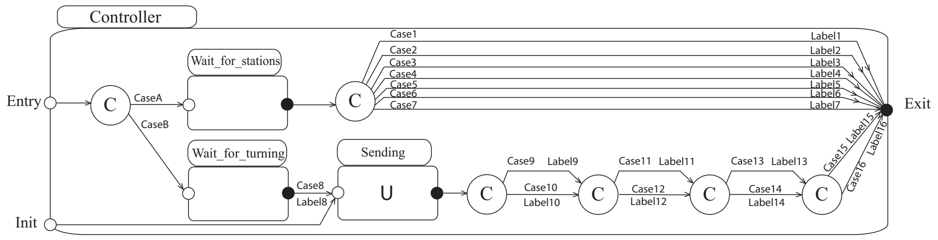


Figure 4. The Controller modeled in REMES.

Table 1. System properties and verification results.

Num	System property	Temporal logic formula	Verification result
1	The system should be free from deadlocks.	A[] not deadlock	Satisfied
2	A product must be clamped when drilled.	A[] Driller.Driller_moving_down imply Driller.drill_ clamp==locked	Satisfied
3	The table should never turn when one of the stations is operating.	A[] (Turntable.Turn1 or Turntable.Turn2) imply (Loader.Idle and Unloader.Idle and Tester.Idle and Driller.Idle)	Satisfied
4	Processing five products should never take more than 25 seconds (assuming at most one failed drilling).	A[] (not loaded_failed and time>25 and failed_products$\leq$1) imply processed_products$\geq$5	Satisfied
5	What is the minimum energy consumption for processing five products?	E$\langle\rangle$ (processed_products==5)	14 300 units

from the stations, and one waiting for the turntable to finish turning.

The submode Wait_for_turning is exited when the turned message arrives. Depending on the current state of the four slots, messages are sent out to the respective station. This is managed by the four conditional connectors and the guards Case9, ..., Case16. For example, the load message is only sent if the first slot is empty, and the drill message is only send if the second slot is occupied. The local variables signal_loader etc. are used to keep track of what messages were sent. When all messages are sent, the history variable System[Controller] is assigned the value Wait_for_stations. Thus, the *Controller* will continue executing in that submode when reentered.

In submode Wait_for_stations, the *Controller* waits until it receives a reply to one of the messages sent. Since this is a non-lazy mode, it must be exited as soon as the guard on one of the outgoing discrete actions (Case1, ..., Case7) is satisfied. If the message carry a value (which is the case for loaded and tested), it is used to update the state of the corresponding slot. When all messages have been received, the message turn is sent to the *Turnable*, and the history variable is set to Wait_for_turning before exiting, meaning that the execution will be resumed in that submode.

3.3 Analysis

We have analyzed the model of the turntable system, transformed into a network of PTA models, in UPPAAL CORA[2]. Currently, the semantic translation from REMES to PTA is done manually, as described in [12], although ideally this step should be fully automated.

After having provided UPPAAL CORA with the PTA model of the turntable system, the last step to verify the system design is to formulate the desired system requirements as temporal logic formulas. Table 1 lists few representative system requirements together with their temporal logic formulas and verification results. Property 1 is a generic safety property, specifying the absence of a system deadlock, i.e., the system cannot come to a state from which it cannot continue operating. The turntable system is verified to be deadlock free. The next step is to verify that it satisfies the functional system requirements, here represented by properties 2 and 3. Properties 4 and 5 are examples of extra-functional properties, addressing time and resource usage, respectively.

[2]UPPAAL CORA web page: `www.cs.aau.dk/~behrmann/cora`

3.4 Packaging as components

The activities illustrated in the above sections concern various functional and extra-functional properties of the components, such as architectural models, REMES behavioral models, PTA models, UPPAAL verification queries and results, among others. To promote their reuse together with the entity they describe, they are all packaged together in a "bundle" of development artifacts, which together constitute a component in the ProCom sense. This packaging is managed by the attribute framework, which provides a common structure to attributes of different kinds, such as metadata specifying the date when an attribute value was entered or edited.

In the integrated development environment providing support for development with ProCom, called PRIDE[3] the attribute framework is also responsible for registering editors by which complex attributes such as REMES models can be viewed and modified. An illustration of the use of the REMES attribute for the ProSys Driller subsystem is presented in Figure 5. Additional attributes could also be specified and used in a similar way, for example attributes corresponding to the timed automata or priced timed automata models.

From the rich REMES model of a component — expressing how e.g., the resource usage changes over time or in response to arriving messages, or how consumption of different resources are related — it is possible to extract isolated extra-functional properties that can be stored as separate attributes. For example, from the model of *Driller* we can derive a bound on the additional power consumed as the result of a received Drill message (the bound is $\mathsf{tdrill2} * \max(\mathsf{eng\_clamp}, \mathsf{eng\_drill})$). Albeit very simple compared to the full REMES model, a MaxEnergy attribute attached to the Drill input port provides useful information about the component and could serve as input to other analysis techniques.

Ideally, any analysis result from a component analyzed in isolation should be stored as an attribute of that component. In the turntable case, the second property in Table 1 holds for the *Driller* subsystem regardless of how the rest of the system behaves. The property could be packaged as a reusable attribute of the *Driller* subsystem. However, the details of how such attribute should be specified are yet to be elaborated.

4 Related work

Few component models incorporate component extra-functional behavioral aspects (e.g. timing, resource usage, etc.) in their frameworks. When they do, they generally provide support for a predefined subset of extra-functional properties and are often intended for research purpose e.g. Palladio [3], BIP [2] or PECT [7].

In these component models, two ways of integrating behavioral models can typically be found: either the behavioral model is an intrinsic part of the component, as in BIP, or it is placed along the component, or the system, as in Palladio, for instance. Our approach positions itself in the middle: the behavioral model is placed alongside the components, but it also is an intrinsic part of the component specification, via the attribute framework. As different from BIP, our approach allows one to attach behavioral models not only to components, but also to individual services, for example. In comparison to Palladio, which is mainly checked by simulation, we use formal behavioral models that allow formal verification of behavior, in addition to simulation, hence increasing the level of trust in the functional and extra-functional behavior of components and systems. Moreover, we also facilitate model reuse, since the REMES behavioral models are part of the structure that constitutes a component. An alternative solution consists in using analytical interfaces jointly with a reasoning framework to perform property predictions such as in BlueArX [8] and PECT [7].

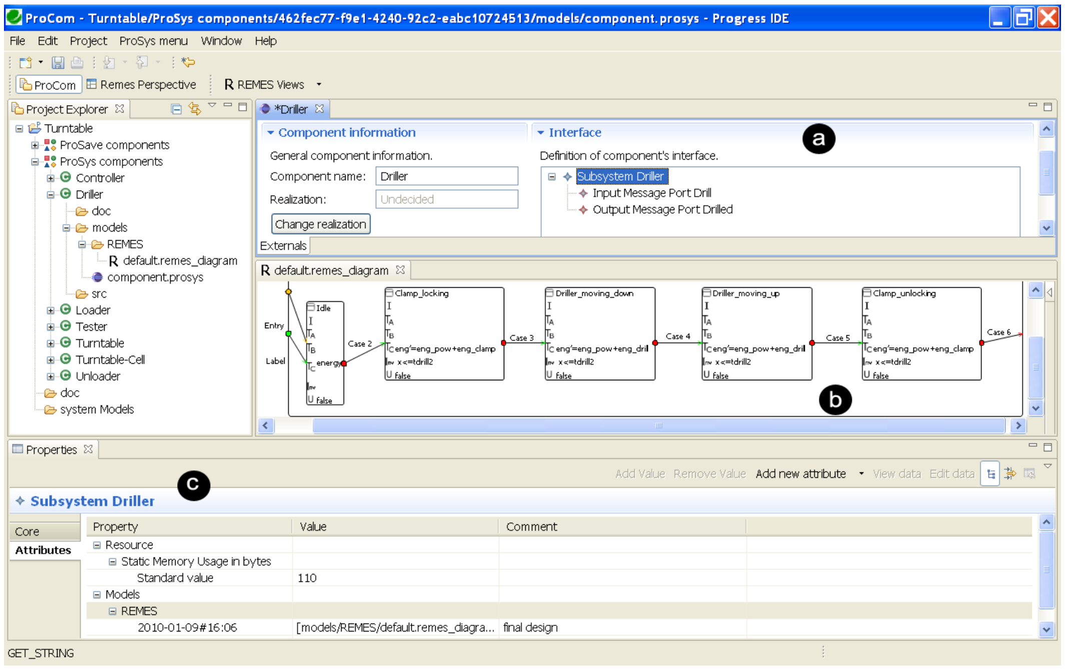
Model-driven development is also gaining interest for early design and analysis of embedded systems, due to automated environments, such as MathLab Simulink-Stateflow [9], and the development of the UML profile for Modeling and Analysis of Real-time and Embedded systems, (MARTE) [11]. In contrast to a component-based approach, this methodology is not centered around the notion of components, and does not focus on reusability.

5 Conclusions and future work

We have presented a component-based approach for modeling both the architecture and behavior of distributed embedded systems. The architectural aspect is modeled according to the component model ProCom, and the behavior of individual components is modeled in REMES, where functionality, timing and resource usage can be addressed together. Transformations of REMES models into timed automata or priced timed automata allow for model-checking of various properties, locally or at system level. By connecting behavioral models to individual components, via a general attribute framework, we address the important problem of model reuse.

Future work includes exercising the scalability of REMES and associated analysis techniques. Instead of generating a timed automata model of the entire system, compositional reasoning could be used to prove global system properties out of individual subsystems, or subsystem clus-

[3]PRIDE and REMES editors are available at `http://www.idt.mdh.se/pride` and `http://www.fer.hr/dices/remes-ide`, respectively.



Component information

General component information.

Component name: Driller

Realization: Undecided

Change realization

Externals

Interface

Definition of component's interface.

Subsystem Driller
- Input Message Port Drill
- Output Message Port Drilled

Subsystem Driller

Property	Value	Comment
Resource		
Static Memory Usage in bytes		
Standard value	110	
Models		
REMES		
2010-01-09#16:06	[models/REMES/default.remes_diagra...	final design

Figure 5. Screenshot from PRIDE with (a) the ProSys editor, (b) REMES editor, and (c) attribute framework.

ters properties. Another approach will envision developing specialized model checking optimizations, which exploit the topology of the ProSys architecture, similar to the work on UPPAAL PORT [5]. The overall approach should also be further validated by case studies involving real industrial systems.

References

[1] 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. *Proc. of the IEEE*, 8(3):231–274, 1987.

[2] A. Basu, M. Bozga, and J. Sifakis. Modeling Heterogeneous Real-time Components in BIP. In *Proc. of SEFM 2006*, pages 3–12. IEEE, 2006.

[3] S. Becker, H. Koziolek, and R. Reussner. Model-Based Performance Prediction with the Palladio Component Model. *Proc. of the 6th International Workshop on Software and Performance*, 2007.

[4] V. Bos and J. J. T. Kleijn. Automatic verification of a manufacturing system. *Robotics and Computer-Integrated Manufacturing*, 17(3):185–198, 2001.

[5] J. Håkansson, J. Carlson, A. Monot, P. Pettersson, and D. Slutej. Component-Based Design and Analysis of Embedded Systems with UPPAAL PORT. In *Proc. of the 6th International Symposium on Automated Technology for Verification and Analysis*, pages 252–257. Springer-Verlag, 2008.

[6] S. Hissam, G. Moreno, J. Stafford, and K. Wallnau. Packaging Predictable Assembly with Prediction-Enabled Component Technology. Technical Report: CMU/SEI-2001-TR-024, 2001.

[7] J. E. Kim, O. Rogalla, S. Kramer, and A. Haman. Extracting, Specifying and Predicting Software System Properties in Component Based Real-Time Embedded Software Development. In *Proc. of the 31st International Conference on Software Engineering (ICSE)*, 2009.

[8] MathWorks. Simulink. `www.mathworks.com/products/simulink/`, accessed February 2010.

[9] Object Management Group. A UML Profile for MARTE, Beta 1, August 2007. Document number: ptc/07-08-04.

[10] C. Seceleanu, A. Vulgarakis, and P. Pettersson. REMES: A Resource Model for Embedded Systems. In *Proc. of ICECCS 2009*. IEEE, 2009.

[11] S. Sentilles, P. Štěpán, J. Carlson, and I. Crnković. Integration of Extra-Functional Properties in Component Models. In *Proc. of CBSE 2009*. Springer Berlin, LNCS 5582, 2009.

[12] S. Sentilles, A. Vulgarakis, T. Bureš, J. Carlson, and I. Crnković. A Component Model for Control-Intensive Distributed Embedded Systems. In *Proc. of CBSE 2008*, pages 310–317. Springer Berlin, 2008.

[13] A. Vulgarakis, S. Sentilles, J. Carlson, and C. Seceleanu. Connecting ProCom and REMES. Technical Report MDH-MRTC-244/2010-1-SE, Mälardalen University, 2010.