Analyzing Resource-Usage Impact on Component-Based Systems Performance and Reliability

Aida Čaušević Paul Pettersson Cristina Seceleanu

Mälardalen Research and Technology Centre (MRTC)
Mälardalen University, 72215 Västerås, Sweden
E-mail: {aida.delic, paul.pettersson, cristina.seceleanu}@mdh.se

Abstract

An early prediction of resource utilization and its impact on system performance and reliability can reduce the overall system cost, by allowing early correction of detected problems, or changes in development plans with minimized overhead. Nowadays, researchers are using both academic and commercial models to predict such attributes, by measuring them at earliest stages of system development. In this paper, we give a short overview of existing prediction models for performance and reliability, targeting popular component-based frameworks. Next, we describe our own approach for tackling such predictions, through an illustration on a small example that deals with estimations of energy consumption.

1. Introduction

In most component-based system development efforts, a great deal of time is spent on ensuring that the functional requirements are being properly implemented, while performance and reliability requirements are given a lower priority [16]. The premise of this paper is that performance and reliability, when added to functionality, constitute a necessary and complete set of metrics for reducing the development cost of complex component-based systems, be they service-oriented or embedded systems.

As the degree of using existing hardware and software resources affects quality attributes like performance and reliability in particular, there is a strong need of system models on which dedicated prediction methods can be applied, as early as possible in the development cycle. Such methods should be able to estimate how various changes in component-wise resource utilization impact on the respective response times (performance metric) or/and number of software faults and errors (reliability metric). The predictions would then later guide the designer towards potential redesigns, e.g., in case the system's resource utilization nears upper thresholds of performance criteria.

Most of the component-based frameworks (CBFs) for system design rely on methods for estimating either performance or reliability changes under given resource utilization scenarios. However, few of the approaches can deliver predictions for any possible system behavior. Most of them cover subsets of behaviors, by using simulation or lightweight formal methods.

The goal of this paper is twofold: first, we briefly review some of the most significant component models and underlying approaches for analyzing the dependency between resource consumption, performance and/or reliability attributes; second, we show how can formal verification techniques, in particular model-checking, be used to predict the performance and reliability of a small real-time, distributed system, modeled as a *priced timed automata* [4]. The intention is to describe a way of carrying out a significant number of experiments, without increasing the system's development cost.

2. Working Example: A Real-time Multiprocessor System

Let us consider a simple distributed system made of 5 components, out of which C0, C1, C2 will be mapped, on a selected target platform, onto 3 real-time tasks that have to execute on 2 available processors, assumed to be components CPU0, CPU1. Moreover, a task component Ci can be executed on just one available processor at any point of time, and it cannot be split in more jobs and executed on both processors. Except for component C1 that can only be executed on processor CPU1, the other two task components can be executed on any available processor.

The tasks are characterized by attributes like cpu_type, computation_time, and deadline, and are assumed to be independent and non-preemptible. The attributes specify the type of CPU on which each component is allowed to execute, the required computation time, and the relative deadline, respectively. The target system abstraction, that is the CPUs, have the attributes type and frequency, denoting the respective CPU's type and speed, respectively.

Assuming a component-based system model, our analysis goal is to select both the best mapping of the task components onto the available processors, and the optimal sequence of execution such that all tasks will meet their deadlines and the energy consumption for each execution is minimized. We recognize here both a *performance* as well as a *reliability* prediction problem of the composition, as follows:

- performance: minimize the energy consumption of each task, for each execution, given the fact that the consumed energy is directly proportional to the task's execution time;
- reliability: minimize the number of software errors, that is, number of times the tasks fail to meet their deadlines, respectively.

Here, the identified resources are energy and the computation resources, that is, the processors.

3. Quality Prediction in Current CBFs

In this section, we will give a brief overview of the current prediction techniques for performance and/or reliability, for some of the most popular state-of-the-art component models, if possible, in the context of the above example. Concretely, we will look at Palladio Component Model, Klaper, SOFA, Koala, Robocop, and BIP.

3.1. Palladio

Palladio Component Model (PCM) represents a domain modeling language used for model-driven performance prediction [3]. The main purpose of introducing PCM is to perform early performance prediction of alternative software architectures. Therefore, the analysis methods are able to calculate metrics like the response time of provided services in a software system, with respect to parametric dependencies within components, and the actual usage profile of a software system. Simulation tools generate simulation source code and scenarios, based on instances of the PCM [11].

In our example's context, assuming that both tasks and processors are each described by a PCM, one could get the following:

- performance: calculate the response time of each task, for various task-processor allocation scenarios and energy-usage profiles;
- reliability: no support provided.

The main advantage of PCM-based prediction methods is that it reduces model complexity by providing models for different component-based software engineering (CBSE) developer roles, which are parameterized with the targeted platform attributes. The disadvantage of the approach is lack of support of reliability prediction techniques, of real-time analysis, and the lower degree of assurance provided by simulation, when compared to full formal verification.

3.2. SOFA

The Software Appliances (SOFA) component model, in its current version SOFA 2 [7], provides a modeling platform for software components, based on a hierarchical model with nested components that are able to communicate over defined interfaces. In terms of prediction, this model comes together with an infrastructure that allows for general component monitoring that gathers performance related information. The performance attributes are fed to the performance modes, and then the gathered information is feedbacked to the component model, whose resource usage level is adjusted accordingly. The process is iterative, stopping when a reasonable trade-off between resource-usage and performance is obtained.

The component behavior is captured by annotating invocations with lists of resource demands. The allocation of resources is described within the deployment model. With SOFA, one can predict how performance attributes change when the application scale changes.

If we assume the working example of section 2, by employing this approach, we can get the following results:

- performance: predict each task's completion time, based on monitoring, and a chosen energy-demand level;
- reliability: not supported.

3.3. KLAPER

KLAPER (Kernel LAnguage for PErformance and Reliability analysis) [9] is a kernel language intended to capture relevant information about non-functional attributes of component-based systems (CBS), focusing mainly on performance and reliability. To derive meaningful analysis models from design models, one can work within the so-called KLAPER-based transformation framework; the latter accepts as input software design models expressed via heterogeneous notations, and produces as output various performance and reliability models. Assuming that we virtually apply KLAPER-based transformations to our task and processor models, the following can be predicted:

- performance: calculate each task's execution time, as a function of the service speed attribute represented by the processor frequency;
- reliability: calculate each task's probability of failure to meet its respective deadline.

A remarkable feature of KLAPER is that it offers the possibility of a direct transformation from design-oriented into different analysis-oriented notations such as Petri Nets, Discrete Time Markov processes, and Extended Queueing Networks (EQN). Another advantage is that one can associate scheduling policies with a resource, in order to model access control policies. As such, the framework allows for a direct estimation of the resource-usage impact on quality attributes like performance and reliability. The lack of feedback between the analysis step and the design models could be considered as a disadvantage of the approach.

3.4. Koala

Koala [15] is a software component model, introduced by Philips Electronics, designed to build product families of consumer electronics. Resource information is exposed at the component's interface. The *provides* interface defines the operations offered by the component, whereas the *requires* interface defines the operations of other interfaces that the component needs to use. Since in a Koala model all the external functionality that is required by the component needs to go through the "requires" interface, it is somewhat straightforward to estimate the use of the system's resources, such as memory utilization. To estimate a Koala component's static memory consumption, one can assume that a special type of *reflection* provides the interface.

In the context of our example, one could analyze the following:

- performance: assuming that all the components (tasks) of the example require the same amount of memory, and that the latter is specified, yet some of the components need to queue to get memory access, one could analyze how various component configurations can affect task execution and system performance, under different memory availability scenarios. In addition, if one tags the task that uses the largest amount of memory as "slow", one could estimate the number of failures that affect the system budget, and compare the new budget with the actual execution cost, hence capturing performance changes;
- reliability: estimate the number of failures that might occur during the execution of "slow" tasks.

The above technique supports budgeting, that is, the expected values of the resource consumption of non-implemented components can also be accounted for. On the other hand, the approach can be used to estimate the total system performance, in a compositional fashion, only on specific, reduced-size scenarios for which the set of components instantiated in a composition is known before runtime.

3.5. ROBOCOP

The Robust Open Component Based Software architecture for Configurable Devices Project (ROBOCOP) is inspired by the Koala component model. It consists of several models that provide parts of the component information, respectively.

To solve the static memory estimation problem, a scenario-based simulation approach has been introduced [5, 6]. This approach delivers resource estimations for a set of scenarios that represent critical usages/executions of the system. The proposed resource model specifies the predicted resource consumption for all the operations implemented by the services of an executable component. As such, the model contains a number of cost functions that give the operations' costs. There can be multiple cost functions, for each resource. To increase the faithfulness of the prediction, the resources that are claimed and released are specified per operation.

Let us assume that we have the resource-wise and functional behavioral model (written for example in binary code) for the components in our example, and the accompanying application scenario that describes service instances and bindings between the respective components. In such case, we would be able to proceed with the analysis described below:

- performance: for components $C0, C1, C2$ their worst-case response times could be checked against the respective deadlines;
- reliability: compute the number of missed deadlines for $C0, C1, C2$; this result can reflect the reliability of the modeled system.

Muskens and Chaudron are describing a method for run-time resource consumption in multi-task CBS, via a formal approach that allows prediction of dynamic resource consumption [14].

3.6. BIP

The acronym BIP stands for **B**ehavior, **I**nteraction, **P**riority, and is a framework for modeling heterogeneous real-time CBS. Each component is obtained as a superposition of three layers. The lower layer describes the component behavior, the intermediate layer includes connectors that describe component interactions, and the upper layer is a set of priority rules that model scheduling policies for interactions. BIP does not make an explicit distinction between inputs and outputs, such that the global variables can be treated either as inputs or as outputs. Basu et al. [2] give an example of performance evaluation of timed tasks that process events from a bursty event generator, all modeled and executed in the BIP framework.

If we employed BIP to tackle our example, we would get the following results:

- performance: worst-case execution time of $C0, C1, C2$, for any valid task-processor allocation scenario, and scheduling analysis via formal verification.

The advantage of the approach is threefold: (a) it accounts for possible heterogeneity of components; (b) it provides a rigorous, correct-by-construction basis for the study of architectural transformations, and (c) it is supported by formal verification tools for the compositional analysis of performance, or important properties such as deadlock-freedom.

4. Our approach

Our appraoch is based on the SaveComp component technology and its component modeling language SaveComp Component Model (SaveCCM), which have been developed within the SAVE project[1] [8]. The semantics of the core part of the language is given as models of timed automata. Having semantics defined in terms of timed automata, we are able to analyze SaveCCM models within different model-checking tools (e.g. UPPAAL[2]). Recent research on SaveCCM has been performed in the area of embedded control applications of vehicular systems [1, 10]. The electronics in vehicles represents a class of systems where quality attributes, such as reliability and resource usage, have impact throughout the development process. The analysis that has been done with SaveCCM within case studies mainly address these topics.

In this section, we will present our model, which is based on Priced Timed Automata (PTA) theory [4], an extension of Timed Automata (TA) with costs on locations and edges. In PTA, costs are associated with edges, to define the cost of executing a corresponding action transition, and location, to define the cost per time unit of delaying there. The PTA framework provides modeling and analysis of continuous, monotonically increasing consumption of resources, e.g. energy consumption. Since the PTA framework does not provide combined reasoning about monotonic (e.g. energy) and non-monotonic resources (e.g. memory), we will treat the whole amount of required memory as static, allocate the total memory amount at the beginning of each task execution, and model it as a discrete value. This problem can be solved with multi-priced TA [13], which are PTA with multiple cost variables evolving according to given rates for each location. Due to space limitation, we will not describe the model of priced timed automata here, but refer the reader to [4] for a thorough description of the framework.

[1]SAVE project is supported by Swedish Foundation for Strategic Research.

[2]The UPPAAL tool is developed in collaboration between Uppsala University, Sweden and Aalborg University in Denmark. More information is available at http://www.uppaal.com/

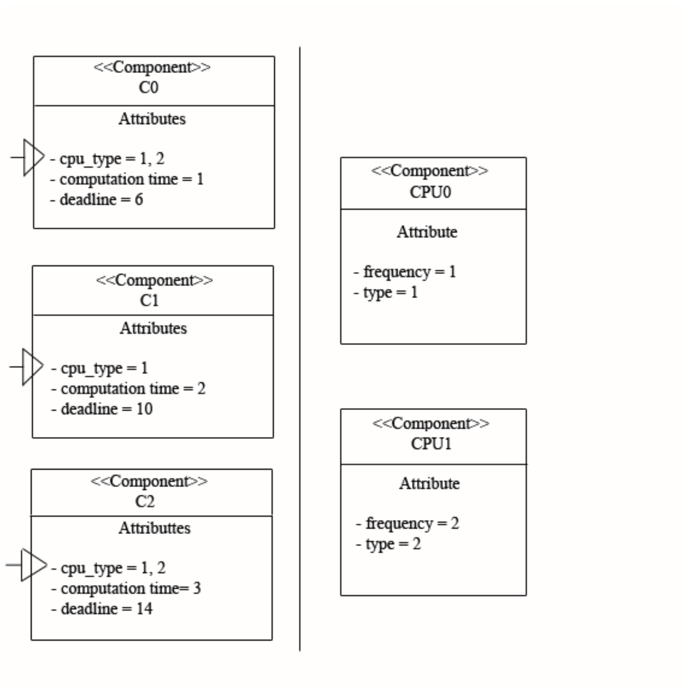

Figure 1. SaveCCM component model

4.1. Example Revisited: Analyzing the Multiprocessor System's Performance and Reliability using UPPAAL

To exemplify our approach, we recall the component-based system presented as our working example in Section 2. The system model is depicted as a SaveCCM-based description in Figure 1. We model the example system as the composition of three real-time tasks T_0, T_1, and T_2, corresponding, in the Save-CCM representation, to $C0$, $C1$, and $C2$, respectively, and two processors P_0 and P_1 describing components $CPU0$ and $CPU1$. Note that tasks are assumed to be independent, that is, their execution do not depend on the state, results, or side effects of the other tasks.

Our first goal is to model non-preemptive multiprocessor

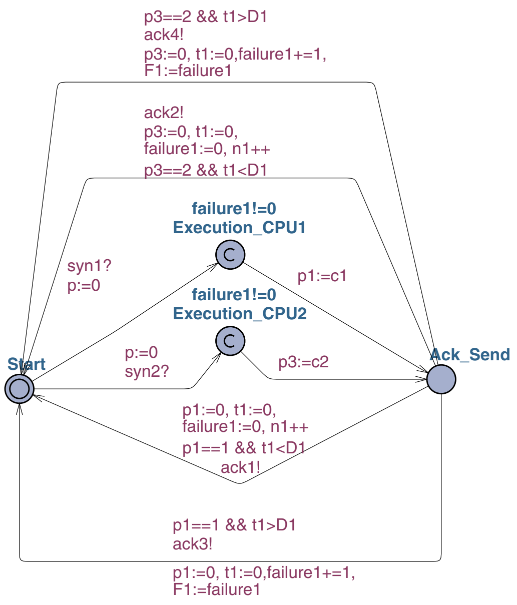

Figure 2. The model of a task.

task scheduling. Tasks (T_i) can be executed in parallel if there are available processing resources (P_j), enabling multiple requests to be served simultaneously. Each task (T_i) is defined by its deadline (D_i). Processors (P_j) are characterized by their period (Per_j) and consumed energy (E_j). We introduce the notion of *task execution time* (ET_i), since the consumed energy is directly proportional to the latter. As such, in a quite simplified form, ET_i can be equated to:

$$ET_i = NoCyc * Per_j$$

where $i \in \{0, 1, 2\}$ is the task identifier, $j \in \{0, 1\}$ is the processor identifier, and N$oCyc$ represents the total number of CPU cycles per task, which we assume known.

The consumed energy, per task, is given by the following equation, also in an abstracted form:

$$E_i = ET_i * w_j * PW_j,$$

where PW_j models CPU power dissipation, which we assume fixed and known. Note that E_i is a weighted function of ET_i and PW_j, where the given weight w_j expresses the relative importance of E_i, which in turn influences the final cost. The accumulated energy consumption is then given as the following cost:

$$c = \sum_{i=0}^{2} E_i$$

Choosing the values of the weights is subjective, depending on both the application and the analysis goals.

When a task execution completes by meeting its deadline, it sends an acknowledgment to some processor to inform that the execution is finished. Our complementary goal is to model a system that would let us predict the total resource usage and its impact on performance and reliability.

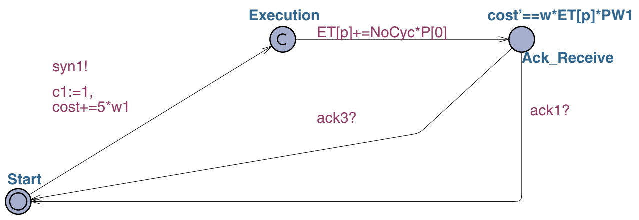

Figure 3. The model of the processing unit.

4.2 PTA Models

We model our example as a collection of five nondeterministic PTA. The PTA (also called processes) communicate using synchronization channels and shared global variables (i.e. variables that can be read and written by all of the processes). The model consists of three automata representing the tasks ($\mathrm{T}_0, \mathrm{T}_1, \mathrm{T}_2$) that are competing for two available automata representing the processors (CPU0 and CPU1).

The model of a task is shown as a PTA in Figure 2. It has two locations: Start and AckSend. The synchronization with an available processor is modeled by using two channels, syn1 (models synchronization with processor P_0), and syn2 (models synchronization with processor P_1). The execution start of a task is controlled by the failure1 variable — a counter (bounded integer) that indicates whether the task failed to meet its deadline or not. The counter is initially set to one, and increased if a failure occurs. If the execution is successful, that is, the deadline is met, the variable is reset to zero. This also indicates that the task is no longer in the ready queue. For each task, the variable pi is assigned the processor number (cj) on which it is currently executing. Depending on the execution result, one of two types of acknowledgment can be sent; in case the task completes successfully, ack1 or ack2 are sent, depending on which CPU the task is synchronizing with; in case the task fails to meet its deadline, ack3 or ack4 are sent.

The PTA model of a processor consists of two locations: Start and AckReceive. In Figure 3, a synchronization channel syn1 is used for synchronization with the tasks present in the ready queue. Variable cj stores the processor number used by task variable pi to identify the task that is being executed on the respective processor. If the execution is successful, acknowledgment ack1 is received by the processor, or ack3 otherwise. The cost of energy consumption is influenced by the assigned weights, execution times, and CPU power dissipation as described previously in this section. The minimum cost of energy consumption

Scenario	Order of execution	Cost
1	(T_0, P_0)-(T_1, P_0)-(T_2, P_0)	15
2	(T_0, P_1)-(T_2, P_0)-(T_1, P_0)	20
3	(T_0, P_0)-(T_2, P_1)-(T_1, P_0)	30

Table 1. Best task mapping with minimum cost.

Sequence of task execution	Ratio
(T_0, P_1)-(T_2, P_0)-(T_1, P_1)	23/10
(T_2, P_0)-(T_0, P_1)-(T_1, P_1)	47/10
(T_2, P_0)-(T_1, P_1)-(T_0, P_0)	20/10
(T_1, P_1)-(T_2, P_0)-(T_0, P_0)	37/10

Table 2. Ratio between number of failures occurred and system executions

is the infimum of the costs of all finite executions from the first to the last state.

4.3 Analysis

The best performance analysis could include finding the best mapping of tasks onto available processors, such that all task deadlines are met, but also the execution order for which the power consumption is minimal. The results are presented as cost values of the computed optimal execution traces. Recall that in our example task T_1 can be only executed on processor P_0.

Usually, the reliability of a system reflects its ability to perform a given function under present conditions, in a specified period of time. Our assumption is that during normal system execution, failures can occur, and this affects directly the overall system reliability. In order to account for failures in our PTA-based model, we analyze the reliability via a ratio between the number of failures occurred during all system invocations, and the number of system invocations. The results are given in Table 2.

We note that the cost is minimum in case when all tasks T_0, T_1, and T_2 are competing for the same processor. The cost value presented in Table 1 shows that the cost is minimal if all tasks are being executed on processor P_0, which is assumed to be "less expensive", than the other one. Of course, cost could be higher if we assigned additional cost for waiting in ready queue. Table 1 presents the cost results assuming all tasks complete successfully. Beside the minimum cost, we also present in Table 1 costs for scenarios in which T_1 has to wait additional time for tasks T_0 and T_2 to complete. Tasks T_0 and T_2 arrive before T_1 to the ready queue and they are allowed to compete for all available CPU resources. In these scenarios, task T_1 is forced to wait in the ready queue, despite the fact that early execution of this task would result in lower cost for the whole system. Clearly, if failures occur during execution, such that the tasks need to be executed more than once in order to complete, the final cost is much higher.

We have noticed that most of the failures occur in situations when two tasks with the greatest and the smallest computation time and deadline (T_2 and T_0, respectively) are competing for the same free processor, and T_2 gains its CPU time (see Table 2). In that case, T_0 has to wait an additional time to start its execution. This problem can be easily solved by including some additional scheduling policy, however this is out of the scope of this paper.

5. Conclusions and Future Work

In this paper, we have briefly reviewed the performance/reliability analysis techniques available in the state-of-the-art component-based frameworks, and their possibility of estimating the impact of changing resource usage on the above mentioned quality attributes. Although extensive work has tackled such problems, the real-time systems area is left less researched. This has motivated us to propose a priced timed automata model-checking approach for component-based systems described in the SaveCCM modeling language that is designed for a real-time and embedded systems. As demonstrated in a small accompanying example, our approach allows for rigorous predictions of performance and/or reliability, depending on the prices of using various resources, such as CPU, memory etc.

In the future, we plan to investigate the possibility of carrying out probabilistic quantitative predictions, by expressing properties to be verified in a probabilistic temporal logic (e.g., PCTL) [12].

Acknowledgments: The authors are grateful to Petr Tuma, Mikael Sjödin, and Tiberiu Seceleanu for their valuable comments on the example used in this paper. This work was partially funded in the context of the Q-ImPrESS research project (FP7-215013) by the European Union under the Information and Communication Technologies priority of the Seventh Research Framework Programme. We also want to express our gratitude to the Swedish national strategic research center PROGRESS, supported by the Swedish Foundation for Strategic Research (SSF) and Mälardalen University, for co-funding this work.

References

[1] M. Åkerholm, J. Fredriksson, K. Sandström, and I. Crnkovic. Quality attribute support in a component technology for vehicular software. In *Fourth Conference*

on Software Engineering Research and Practice in Sweden, October 2004.

[2] A. Basu, M. Bozga, and J. Sifakis. Modeling heterogeneous real-time components in bip. In *SEFM '06: Proceedings of the Fourth IEEE International Conference on Software Engineering and Formal Methods*, pages 3–12, Washington, DC, USA, 2006. IEEE Computer Society.

[3] S. Becker, H. Koziolek, and R. Reussner. Model-based performance prediction with the palladio component model. In *WOSP '07: Proceedings of the 6th international workshop on Software and performance*, pages 54–65, New York, NY, USA, 2007. ACM.

[4] 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.

[5] E. Bondarev, M. R. V. Chaudron, and P. H. N. de With. Compositional performance analysis of component-based systems on heterogeneous multiprocessor platforms. In *EUROMICRO-SEAA*, pages 81–91, 2006.

[6] E. Bondarev, P. de With, M. Chaudron, and J. Muskens. Modelling of input-parameter dependency for performance predictions of component-based embedded systems. In *EUROMICRO '05: Proceedings of the 31st EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 36–43, Washington, DC, USA, 2005. IEEE Computer Society.

[7] T. Bures, P. Hnetynka, and F. Plasil. Sofa 2.0: Balancing advanced features in a hierarchical component model. In *SERA '06: Proceedings of the Fourth International Conference on Software Engineering Research, Management and Applications*, pages 40–48, Washington, DC, USA, 2006. IEEE Computer Society.

[8] J. Carlson, J. Haakansson, and P. Pettersson. SaveCCM: An analysable component model for real-time systems. In Z. Liu and L. Barbosa, editors, *Proceedings of the 2nd Workshop on Formal Aspects of Components Software (FACS 2005)*, volume 160 of *Electronic Notes in Theoretical Computer Science*, pages 127–140. Elsevier, 2006.

[9] V. Grassi, R. Mirandola, and A. Sabetta. From design to analysis models: a kernel language for performance and reliability analysis of component-based systems. In *WOSP '05: Proceedings of the 5th international workshop on Software and performance*, pages 25–36, New York, NY, USA, 2005. ACM.

[10] H. Hansson, M. Åkerholm, I. Crnkovic, and M. Torngren. Saveccm - a component model for safety-critical real-time systems. In *EUROMICRO '04: Proceedings of the 30th EUROMICRO Conference*, pages 627–635, Washington, DC, USA, 2004. IEEE Computer Society.

[11] K. Krogmann. Reengineering of Software Component Models to Enable Architectural Quality of Service Predictions. In R. H. Reussner, C. Szyperski, and W. Weck, editors, *Proceedings of the 12th International Workshop on Component Oriented Programming (WCOP 2007)*, volume 2007-13 of *Interne Berichte*, pages 23–29, Berlin, July31 2007. Universität Karlsruhe (TH).

[12] M. Kwiatkowska, G. Norman, J. Sproston, and F. Wang. Symbolic model checking for probabilistic timed automata. *Inf. Comput.*, 205(7):1027–1077, 2007.

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

[14] J. Muskens and M. R. V. Chaudron. Prediction of run-time resource consumption in multi-task component-based software systems. In *CBSE*, pages 162–177, 2004.

[15] R. van Ommering, F. van der Linden, J. Kramer, and J. Magee. The koala component model for consumer electronics software. *Computer*, 33(3):78–85, 2000.

[16] P. R. Work and J. H. E. (John) Johnson. Risk Management in Computer-Based Systems Development by Use of Performance and Reliability Metrics. In *Proceedings of the 1995 International Symposium and Workshop on Systems Engineering of Computer Based Systems*, pages 367–373. IEEE, 1995.