Monitoring and Stochastic Analysis of Component-Based Control-Systems

Anders Möller[‡ ⋆] Mikael Nolin[‡ ⋆] Ian Peake[†] Heinz W. Schmidt[†]

[‡]MRTC, Mälardalen University, Västerås, Sweden
[†]Monash Univeristy, Melbourne, Australia
[⋆]CC Systems, Uppsala, Sweden

Anders.Moller@mdh.se

Abstract

A software component is a suitable vehicle to introduce advanced analysis techniques in a software-engineering context for embedded control-systems; a feat that has yet to be fully accomplished.

This project is adopting a component-based approach to control-system software development. We are extending and combining methods from disparate disciplines, such as probabilistic reliability predictions, stochastic scheduling analysis and software component technologies. We study theories and methods for probabilistic modelling, analysis, and prediction of control-system software executing in resource-constrained embedded computers.

Combining the behaviour models and the architectural model of a component assembly we are deriving stochastic properties, such as reliability, expected delays, and resource consumption. Using components as the fundamental unit of reuse, we employ run-time monitoring techniques to extract probabilistic models of the component behaviour.

1 Introduction

Developers of embedded real-time control-system face challenges of (*i*) high demands on reliability and performance (*ii*) requirements on lowered product cost, and (*iii*) supporting many configurations, variants and suppliers. Computer systems offer the performance needed for the functions requested, but at the same time product reliability and safety must not suffer. Unfortunately computers and software add new sources of failures, and therefore can cause lessened product reliability. This yields a strong focus on the ability to *model*, *predict*, and *verify* software functionality, reliability, and safety. However, many companies lack the technical support to verify their computer system behaviour (both with respect to functional and extra-functional aspects of the software) - and black-box testing together with manual code inspections are usually the only methods used to "confirm" system functionality and reliability.

Also, in order to keep the software development costs within budget, more and more Original Equipment Manufacturers (OEMs) use sub-contractors (and/or Commercial-Off-The-Shelf (COTS) Components) to develop various parts of their computer system. This increases complexity of system analysis and jeopardises software system trust. Due to the potentially high (economic and/or safety) impact of software failures (e.g., passenger safety in a car) – predictable software becomes increasingly important.

The real-time systems of interest to us are distributed, and therefore require dealing with parallel behaviour (or at least communication of a sequential component with parallel components in its deployment environment). Our work focuses initially on the sequential components and the behaviour of multiple components executing on the same embedded controller. Therefore asynchrony and scheduling issues are inside the boundaries of our project while true parallelism is outside.

2 Project Aims

In this project, we are studying theories and methods for probabilistic modelling, analysis, and prediction of control-system software executing in resource-constrained embedded computers. The overall goal is to come up with methods that will result in (1) a decrease in system life-cycle costs, and (2) an increase in system quality.

Using components as the fundamental unit of reuse, we employ run-time monitoring techniques to observe the systems and extract probabilistic models of the component behaviour [24]. Combining the behaviour models and the architectural model of a component assembly, we are deriving stochastic properties, such as reliability, expected delays, and resource consumption. We are extending and combining methods from disparate disciplines, such as probabilistic reliability-predictions [1, 21] stochastic scheduling-analysis [12, 20, 26], run-time monitoring [2, 24, 25], and software-component technologies [3, 10].

The data obtained from monitoring the components are used to address the following key areas in software engineering of embedded control-systems:

- **Model-extraction**. By monitoring component-based software, information about the component properties can be extracted. This information can be used to describe the components' externally visible properties. These properties provide a basis for trust in components and can be used

for system-level predictions.

- **System-level predictions**. By using models of the component behaviour, key properties such as reliability, timing behaviour and resource consumption can be predicted. Early prediction of such properties can be used to guide system designers and aid in dimensioning hardware resources, hence reduce the development effort and increase quality.
- **System-level testing and debugging**. By monitoring individual components and component interactions, errors can be found and traced. Monitoring can also be used to support replay debugging [25], where erroneous system-executions are recreated in a lab environment to allow tracing of bugs. Testing and debugging of embedded systems are notoriously difficult and time-consuming. Hence, increased support for test and debug has great potential for cost savings.
- **Run-time contract checking**. The run-time monitoring allow surveillance of third party components. Both functional (e.g. range of output values) and extra-functional (e.g. memory usage) properties can be monitored. During acceptance testing, the contract checking is used to validate that a component does not violate its specification. In systems that fail after deployment, logs from the contract checking can be used in post-mortem analysis. The possibility for post-mortem assessment of contract breaches is likely to increase the willingness to deploy third-party components in critical systems, since such assessments can be coupled to financial liabilities.
- **Observability**. Computer systems in general and embedded systems in particular, are infamous for the difficulty of observing their internal behaviour. This has drawbacks throughout the debugging, testing and maintenance phases. Systems that are unobservable become very difficult to analyse and validate. Also after deployment, observability is an important feature, allowing inspection and performance tuning of running systems.

To realize the above goals, this project contains two main strands of research:

✓ Development of monitoring techniques suitable for resource-constrained embedded systems. Monitoring of resource constrained systems requires lightweight techniques to be developed. Also, the number of metrics (and their quality/granularity) should be limited. This presents a trade-off between achieved results of the monitoring and the resources consumed.

✓ Development of techniques to predict system properties from component properties. We focus our research on stochastic methods such as reliability prediction and probabilistic scheduling-analysis. Whether to deploy stochastic or deterministic (worst-case) predictions is a strategic decision for each project. However, the majority of systems (aerospace applications exempted) are not designed for worst-case scenarios; it is not possible to motivate the cost incurred by dimensioning and validating the system for the worst-case scenario. Hence, our focus is on stochastic methods that allow designers to make well founded trade-offs between, e.g., reliability and production cost.

3 Component-Based Development

Component-Based Development (CBD) for resource-constrained embedded real-time systems usually differ from genaral CBD for Internet/office applications, in which components are individually compiled and deployed. At run-time, components are loaded and bindings between component are dynamically created by a middleware. Figure 1 illustrates the typical process when using CBD for *embedded* systems. Systems are developed by assembling components from a repository, or by using newly developed components. In the next step a system generator generates C source code where component bindings are explicitly represented and connected. Finally the C code is compiled and the system executed. Upon the detection of bugs or undesired system properties, the system designer goes back and modifies the system.

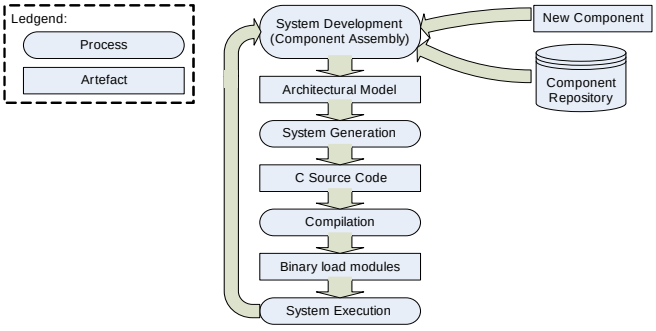

Figure 1. Traditional CBD Processes for Embedded Systems

Using the results of this project, a much more attractive process can be obtained. Figure 2 illustrates the new possibilities with highlighted activities. Firstly, using analysis, early assessments about system properties can be made, also – if the system is redesigned undesired properties can be detected. Secondly, the system generator can be augmented to automatically insert the instrumentation code needed to perform run-time monitoring. Finally, dynamic properties for the components are extracted during run-time. These properties are then stored together with the components in the repository and can be used in subsequent development cycles.

4 Survey of the Field

This project will draw upon results from several established research fields.

4.1 Component-based Development

During the last decade, tremendous advancements have been made in component-based development (CBD) for desktop- and Internet-applications. A set of commercially available techniques has transformed the way commodity software is developed. Some of the most well known techniques are Microsoft's COM and .NET, SUN's Enterprise Java Beans, OMG's Corba Component Model (CCM). However, for embedded systems no readily available technique exist [3, 16]. Within the research community a plethora of component models and component techniques for embedded systems exits, see e.g. Nolin *et al.* for a survey [19].

Such techniques, like the well known Koala component model from Phillips [28], often strive to generate resource-conservative systems. However, as outlined in section 2, designers of control-systems often have to pay attention to other properties, such as reliability and timeliness. To this end, many projects have come up with component models that should support analysable systems, e.g. [10, 23, 31, 33]. All these techniques are based upon static, worst-case analysis of the system. Hence, they are dependent on the availability of models that bound the behaviour of the components. However, methods to statically and safely bound properties like execution-time, memory usage, and reliability are not readily available[1]. Also, as discussed in section 2, in most projects it is not feasible to design the system to sustain worst-case scenarios. Unfortunately, no component technologies explicitly address the issue of providing statistical metrics for system behaviour.

Crnkovic and Larsson provides a good overview of the problems needed to be tackled when employing CBD for embedded systems [4]. Möller *et al.* [17], and Hammer *et al.* [7] describes, specifically for embedded systems, what are the requirements to be met by a component technology.

4.2 Run-Time Monitoring and Model Extraction

In [24] we give an overview of the work done on monitoring software-components. Typically, existing component-related monitoring techniques are not suitable for our goals of supporting test, debug, and model extraction (the exception being PECOS [33]). The most relevant work in this area is:

- Monitoring of real-time systems is covered by Chodrow et al. [2]. And monitoring of distributed real-time systems is presented by Tokuda et al. [27]. The generally conclusion is that it is necessary to leave the software probes in the target system at all time in order to eliminate probe-effects.
- Monitoring to support replay debugging [25], where logs from a system are used to exactly recreate the execution of the system. Typically this is used to recreate the faulty execution leading up to a failure of the system.
- Stochastic model extraction has been proposed by Wall *et al.* [9, 29]. They are tackling the problem of extracting models for complex real-time systems, not using any information about architectural style or software structure.
- The PECOS component technology [33] stands out by being, to the best of our knowledge, the only technology that proposes a general approach to monitoring in order to extract component properties. PECOS enables support for instrumenting components during run-time, with respect to, e.g., periodicity, memory consumption, and execution time. However, in PECOS, only worst-case figures are collected.

4.3 Reliability Predictions for Component-Based Software

Mitzenmacher [15] gives a good introduction to probabilistic analysis techniques. The stochastic behaviour of software can be modelled by using Markov chains, where the states are defined by the software components. Markov processes are useful when modelling random behaviours of software, e.g., faults remaining at time t or failures experienced by the time t.

The most relevant work in the area of reliability predictions is:

- Littlewood's *"Littlewood model"* [13, 14] is the first architecture-based software reliability model. It is designed for continuously running systems, and is based on continuous time Markov chains. Several extensions have been made, e.g., by Ledoux [11] whom introduces failure-process effects of the execution, and delays in recovering after a failure.
- Cheung [1] introduced a user-oriented model to predict reliability by using a Markov process to represent the control-flow between different software modules. Reussner *et al.* [21, 22] extended this work to component-based architectures by introducing contractually specified component contracts. Cheung's work is further extended by, e.g., Wang *et al.* [32] whom presents a prediction model to estimate architecture-based software reliability for different architectural patterns.
- Musa *et al.* [18] gives an overview of software reliability in general, and presents a generalised approach of reliability predictions using Markov processes. Musa *et al.* also put forward a classification of finite-failure category models, for Markovian models.

4.4 Stochastic Schedulability-Analysis of Real-Time Systems

Different from traditional (worst-case) scheduling-analysis, this research aims at predicting statistical properties of real-time systems by using stochastic models of the executing tasks. Stochastic schedulability-analysis has not yet been addressed for component-based systems. However, there are some initial results within the real-time computing area:

[1] Even though good progress in static analysis areas such as execution time analysis has been made the last few years, one can hardly claim that such methods are readily available to the general public.

- Tia *et al.* [26] present Probabilistic Time Demand Analysis (PTDA) which is restricted to systems that are using fixed priorities. Gardner *et al.* [6] present Stochastic Time Demand Analysis (STDA) which is better than PTDA in the sense that it can cope with general deadlines.
- Another group of stochastic analysis methods is the Real-Time Queueing theory by Lehoczky [12]. Real-Time Queueing theory can provide stochastic guarantees. However, it requires high traffic load, thus not suitable for a general system configuration. Approaches using Markov processes to scheduling analysis has been proposed by Diaz et al. [5].
- Simulation-based predictions of execution-times has also been proposed [8, 30]. While difficult to quantify the reliability of the results, these approaches have the potential to deal with highly complex models and system behaviour.

5 Research Topics

This project draw upon existing results from several research directions. The main challenges will include (1) to identify suitable techniques, and (2) modify these techniques to suit component-based development of embedded systems. For each of the research directions we list some of the key research questions addressed within this project.

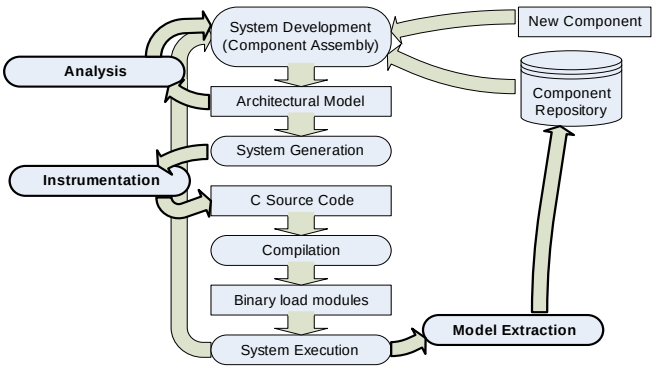

Figure 2. Our proposal to CBD for Embedded Systems

5.1 Software Component Monitoring

Current approaches rely on software instrumentation to produce the logs. Part of this instrumentation can be placed from operating system functions, which minimises the manual effort for instrumentation. However, not all data needed for the logs can be extracted in operating system calls, and manual instrumentation of application code is required. Furthermore, this manual instrumentation requires expert knowledge of the application code and about replay debugging, to make sure that all the needed data is collected.

Using a component-based approach gives us the opportunity to automatically add code instrumentation. This is possible if the component model makes explicit all data flows and state variables, and this is the case for the component models considered in this project [10].

Some specific questions studied within this project are:

- Monitoring embedded systems require resource conservative monitoring-techniques. Are current techniques good enough? Can they be modified to better suit embedded real-time systems?
- Since resources are constrained, the number of metrics (and their quality/granularity) should be limited. This presents a trade-off between achieved results of the monitoring and the resources consumed. A key problem in this project is to identify the metrics that have the highest beneficial impact on the software engineering process.
- Ideally, metrics should be collected and refined throughout the life-cycle of a component. However, resource limitations may hinder this ideal solution since different metrics could be more important during different phases. This presents a trade-off of which metrics to obtain during what stage. A goal of this project is to identify which metrics are most important during what stages.
- The problem that arises with monitoring is the large number of executions necessary to establish a reasonable statistical confidence. Due to the complex behaviour of software components, traditional statistical methods to estimate the statistical validity, such as confidence intervals, may have to be used together with more domain specific validity metrics. We will investigate to what extent traditional statistical methods are suitable and, if needed, try to find methods to complement them.

5.2 Reliability Predictions

We focus on reliability predictions using Markov models, as described in, e.g., [1, 21]. However, such methods require that a system has a well-defined start and end state. Unfortunately, this is not the case for reactive control-systems. In addition, the Markov behaviour of software components has to be mapped to the execution histories and their stochastic properties. This requires blending automata models with Markov chains.

Also, since reusable components are intrinsically open - they have to interact with a range of different external components in different deployment environments, each exhibiting different extra-functional properties. A single fixed or constant model for their behaviour or extra-functional properties results in hopelessly inaccurate predictions. This is because the extra-functional properties of interest are *system properties* emerging from a system model (deployment environment) that is the result of composing (partial) component-level models of those properties developed (in isolation) and reused with the component themselves.

Our prior research, e.g. [21, 22], has shown that reasonably accurate prediction of these system properties can be achieved by parameterisation. Component-level, extra-

functional, models are variable with formal parameters ranging over different behaviours or different extra-functional variations.

Interesting research questions include:

- How can the modelling techniques be adapted/extended in order to fit reactive systems, without start and end states? One possible solution could be to introduce additional super-states, defining the start and end states. This, however, needs to be combined with a way of finding recurrent states in the reactive control-system, i.e. system states that will re-occur during execution.
- To facilitate architecture-based reliability predictions, software components need to be equipped with reliability figures. These figures depend on the context in which the component is used (often called the components usage-profile), e.g., what input ports of the component are used and what output is required. The usage-profile of a specific component in a certain application, of course, has influence on the reliability. What type of reliability measure best suits the control-system domain, and what does this imply in terms of changes in the analysis techniques?
- When the component is deployed with its extra-functional model, the parameters are actualised and the model is instantiated to reflect more accurately key properties of the environment components it relies upon, such as the reliability of services it requests from the underlying system or another used component.
- Is it possible to extend the parameterisation techniques in order to enhance the prediction accuracy. Is it possible to extend the parameterisation technique to other extra-functional models, such as schedulability.
- One basic idea is to transfer the architecture description of the component-based system into a Finite State Automata (FSA), and use reliability figures of the components to represent system reliability. Is it possible to adapt this method, in order to be more fine-grained, e.g., by representing also the actual components with FSAs? In this case, reliability methods can be used to find the weakest part of each of the used components.
- By using component impact analysis, software developers can be guided to put focus on the part of the application that is most crucial in terms of reliability and usage. How can the architecture-based reliability predictions, using a Markov chain representation, be extended in order to guide control-system developers during the design-phase?

5.3 Stochastic Schedulability Analysis

Traditional methods for stochastic schedulability-analysis assume that execution-times can be described by known probability functions. However, such a description is often misleading since execution times cannot take on arbitrary values. Typically, execution times are clustered around a small set of "probability peaks" which is not easy to represent with general probability functions.

On the other hand, if the representation of the execution-times and their probabilities is too detailed, statistical analysis becomes infeasible due to the combinatorial explosion in the number of possible combinations of execution times.

- We investigate histogram-based schedulability-analysis. A histogram can be adapted to provide the desired level of detail, thus it gives the possibility to reduce complexity of the analysis.
- When performing stochastic schedulability-analysis there are two major sources of errors: (1) errors in the models used (e.g. due to not monitoring the component long enough), and (2) errors due to simplifications/abstractions made during the analysis. The effect of both these errors needs to be quantified and bounded. This means that it is not enough to calculate the result of the schedulability analysis, but also a level of confidence that account for both error source is needed.

6 Conclusion

This project will provide novel models, theories and techniques to monitor and predict the behaviour of embedded, resource constrained, control-systems. This class of systems constitutes an increasing fraction of value in many products, such as in vehicles and automation robotics.

We fuse scientific methods from disparate disciplines, such as probabilistic reliability-predictions, stochastic scheduling-analysis, run-time monitoring, and software-component technologies. This will result both in novel methods and extensions to existing, well established, methods.

This project takes a unique approach:

✓ compared to other component-technology projects, we do not focus on developing our own component technology. Rather we focus on the ability to use the architectural- and component-models to facilitate system-level analysis. Our goal is to automate the whole process, thus we do not introduce additional burdens for the software-engineers.

✓ compared to other analysis-techniques project, we do not focus on generally applicable methods. Rather we constrain our research to methods suitable for component-based systems. Since component models provides strict rules on component-interaction this significantly simplifies our task.

References

[1] R. C. Cheung. A User-Oriented Software Reliability Model. *IEEE Transactions on Software Engineering*, 6(2):118–125, 1980. Special collection from COMPSAC 1978.

[2] S. Chodrow. Run-time monitoring of real-time systems. In *Proc. of IEEE 12^{th} Real-Time Systems Symposium*, pages 74–83, December 1991. San Antonio, USA.

[3] I. Crnkovic. Componet-Based Approach for Embedded Systems. In *Proceedings of 9^{th} International Workshop on Component-Oriented Programming*, June 2004. Oslo, Norway.

[4] I. Crnkovic and M. Larsson. *Building Reliable Component-Based Software Systems*. Artech House publisher, 2002. ISBN 1-58053-327-2.

[5] J. L. Díaz, D. F. García, K. Kim, C. G. Lee, L. LoBello, J. M. López, S. L. Min, and O. Mirabella. Stochastic Analysis of Periodic Real-Time Systems. In *Proceedings of the 23^{rd} IEEE Real-Time Systems Symposium*, pages 289–300, December 2002. Austin, Texas, USA.

[6] M. K. Gardner and J. W. Liu. Analyzing Stochastic Fixed-Priority Real-Time Systems. In *Proceedings of the 5^{th} International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, March 1999.

[7] D. K. Hammer and M. Chaudron. Component-Based Software Engineering for Resource-Constraint Systems: What are the Needs? In *Proceedings of the 6^{th} Interntational Workshop on Object-Oriented Real-Time Dependable Systems (WORDS)*, January 2001. Rome, Italy.

[8] H. Hansson, T. Nolte, C. Norström, and S. Punnekkat. Integrating Reliability and Timing Analysis of CAN-based Systems. *IEEE Transaction on Industrial Electronics*, 49(6), 2002.

[9] J. G. Huselius and J. Andersson. Model Synthesis for Real-Time Systems. In *Proc. of the 9^{th} European Conference on Software Maintenance and Reengineering (CSMR)*, pages 52–60, March 2005.

[10] M. Åkerholm, A. Möller, H. Hansson, and M. Nolin. Towards a Dependable Component Technology for Embedded System Applications. In *Proceedings of the 10^{th} IEEE International Workshop on Object-oriented Real-Time Dependable Systems (WORDS05)*, February 2005. Sedona, Arizona, USA.

[11] J. Ledoux. Availability model of modular software. *IEEE Trans. on Reliability*, 48(2):159–168, 1999.

[12] J. P. Lehoczky. Real-Time Queuing Network Theory. In *Proceedings of the 18^{th} IEEE Real-Time Systems Symposium (RTSS'97)*, pages 58–67, San Francisco, CA, USA, December 1997. IEEE Computer Society.

[13] B. Littlewood. A Reliability Model for Systems with Markov Structure. *Applied Statistics*, 24(2):172–177, 1975.

[14] B. Littlewood. Software Reliability Model for Modular Program Structure. *IEEE Transactions on Reliability*, 28(3):241–246, 1985.

[15] M. Mitzenmacher and E. Upfal. *Probability and Computing - Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2004. ISBN 0521835402.

[16] A. Möller, M. Åkerholm, J. Fredriksson, and M. Nolin. Evaluation of Component Technologies with Respect to Industrial Requirements. In *Euromicro Conference, Component-Based Software Engineering Track*, August 2004.

[17] A. Möller, J. Fröberg, and M. Nolin. Industrial Requirements on Component Technologies for Embedded Systems. In *Proceedings of the 7^{th} International Symposium on Component-Based Software Engineering*. 2004 Proceedings Series: Lecture Notes in Computer Science, Vol. 3054, May 2004. Edinburgh, Scotland.

[18] J. D. Musa, A. Iannino, and K. Okumoto. *Software Reliability: Measurement, Prediction, Application*. McGraw-Hill, Series in Software Engineering ant Technology, 1987. ISBN 0-07-044093-X.

[19] M. Nolin et al. Component-Based Software for Embedded Systems - A Literature Survey. Technical report, MRTC Report No 104, ISSN 1404-3041, ISRN MDH-MRTC-104/203-1-SE, Mälardalen Real-Time Reseach Centre, Mälardalen University, June 2003. Västerås, Sweden.

[20] T. Nolte, A. Möller, and M. Nolin. Using Components to Facilitate Stochastic Schedulability. In *Proceedings of the 24^{th} Real-Time System Symposium – Work-in-Progress Session*. IEEE Computer Society, December 2003. Cancun, Mexico.

[21] R. H. Reussner, I. H. Poernomo, and H. W. Schmidt. Reasoning about Software Architectures with Contractually Specified Components. *Component-Based Software Quality. LNCS 2693, Springer-Verlag*, pages 287 – 325, 2003.

[22] H. W. Schmidt. Trustworthy components: compositionality and prediction. *Journal of Systems and Software, Elsevier Science Inc*, 65(3):215–225, 2003.

[23] J. A. Stankovic. VEST – A toolset for constructing and analyzing component based embedded systems. *Lecture Notes in Computer Science*, 2211:390, 2001.

[24] D. Sundmark, A. Möller, and M. Nolin. Monitored Software Components – A Novel Software Engineering Approach –. In *Proceedings of the 11^{th} Asia-Pasific Software Engineering Conference, Workshop on Software Architectures and Component Technologies*, November 2004. Pusan, Korea.

[25] H. Thane, D. Sundmark, J. Huselius, and A. Pettersson. Replay Debugging of Real-Time Systems Using Time Machines. In *Proceedings of Parallel and Distributed Systems: Testing and Debugging (PADTAD)*, pages 288 – 295. ACM, April 2003.

[26] T. S. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L. C. Wu, and J. S. Liu. Probabilistic Performance Guarantee for Real-Time Tasks with Varying Computation Times. In *Proceedings of the 1^{st} IEEE Real-Time Technology and Applications Symposium (RTAS'95)*, pages 164–173, Chicago, IL, USA, May 1995. IEEE Computer Society.

[27] H. Tokuda, M. Kotera, and C. Mercer. A Real-Time Monitor for a Distributed Real-Time Operating System. In *Proceedings of ACM Workshop on Parallel and Distributed Debugging*, May 1988. Madison, USA.

[28] R. van Ommering et al. The Koala Component Model for Consumer Electronics Software. *IEEE Computer*, 33(3):78–85, March 2000.

[29] A. Wall, J. Andersson, J. Neander, C. Norström, and M. Lembke. Introducing Temporal Analyzability Late in the Lifecycle of Complex Real-Time Systems. In *Proc. of the 9^{th} International conference on Real-Time Computing Systems and Applications (RTCSA'03)*, 2003.

[30] A. Wall, J. Andersson, and C. Norström. Probabilistic Simulation-based Analysis of Complex Real-Times Systems. In *6^{th} IEEE International Symposium on Object-oriented Real-time distributed Computing*, May 2003.

[31] K. C. Wallnau. Volume III: A Component Technology for Predictable Assembly from Certifiable Components. Technical report, Software Engineering Institute, Carnegie Mellon University, April 2003. Pittsburg, USA.

[32] W.-L. Wang, Y. Wu, and M.-H. Chen. An Architecture-Based Software Reliability Model. In *Proceedings of the 1999 Pacific Rim International Symposium on Dependable Computing*, 1999. Hong Kong, China.

[33] M. Winter, T. Genssler, et al. Components for Embedded Software – The PECOS Apporach. In *The 2^{nd} International Workshop on Composition Languages, in conjunction with the 16^{th} ECOOP*, June 2002. Malaga, Spain.