Using Components to Facilitate Stochastic Schedulability Analysis

Thomas Nolte Anders Möller Mikael Nolin
Mälardalen Real-Time Research Centre
Deptartment of Computer Science and Engineering
Mälardalen University, Västerås, SWEDEN

E-mail: thomas.nolte@mdh.se

Abstract

In this work-in-progress paper we present how Component Based Software Engineering (CBSE) may be used to facilitate stochastic schedulability analysis of embedded real-time systems, by providing realistic models of execution time distributions.

We present our ongoing work regarding the usage of Execution Time Profiles (ETPs) to represent the timing behaviour of real-time components. These ETPs are to be used in a tool for stochastic schedulability analysis of embedded real-time systems. The tool is intended for real-time engineers to make cost-reliability trade-offs by dimensioning hardware resources in a cost efficient way to achieve the reliability goals.

1 Introduction

Component Based Software Engineering (CBSE) is today recognised as a promising technology to achieve software reuse, efficient software development, and reliable software systems [4]. Traditionally, a *software component* is used to encapsulate some functionality. That functionality is accessed though the *interface* of the component. However, current research is addressing issues on how to associate extra-functional attributes (such as execution time, memory consumption, and reliability) to the components. These extra-functional attributes are intended to be used by tools that analyse systems built using components. For example, in order to perform *deterministic schedulability analysis* the components should be annotated with their Worst-Case Execution Time (WCET) (other information, such as the mapping of components to tasks is also needed, but that is typically not regarded as extra-functional information).

In the real-time community large efforts has been spent on schedulability analysis. Typically, deterministic schedulability analysis is concerned with predicting the worst case behaviour of a system. Several pessimistic assumptions are usually made regarding the system behaviour. Such assumptions include: each task execute for its WCET, the phasing between tasks is the worst possible (the critical instant assumption), each task executes with its maximum allowed rate, each task experience its maximum blocking on shared resources, etc.

Naturally, schedulability analysis made under these assumptions leads to results which are exaggeratedly pessimistic for the normal operation of the system. For systems with high criticality, such pessimism may be warranted. However, for many systems the predictability achieved by performing a schedulability analysis is desired, whereas the cost for dimensioning the hardware for the extreme worst case is not justified. For these types of systems a *stochastic schedulability analysis*, giving some quantifiable measure of the systems' expected performance, could allow system designers to make well founded trade-offs between system reliability, or Quality of Service (QoS) and the amount of system resources required by the system (cost versus quality trade-off). That is, a designer could decide to allow some probability of system timing-failures and in return be able to use cheaper hardware, or the other way around, the designer could choose the cheapest hardware that makes the system fulfil its reliability or QoS requirements.

The problem with existing stochastic schedulability analysis techniques is typically that they make assumptions on the system that are unrealistic. Most notably, the assumptions on execution time are that they are described using known probability distribution functions. However, the execution time of software is generally not very well described by probability distribution functions. Typically, a piece of software has a small set of "probability peaks" around which most execution times are gathered. Furthermore, exception and error handlers are usually executed with very low probability, but yet, when exception handling takes place the execution time may be significantly longer than normal. Hence, representing execution time probabilities with simple distribution functions do not necessary give good correlation between the representation and real exe-

cution times.

In this paper we will present how CBSE may be used to facilitate stochastic schedulability analysis using realistic models for execution time distributions. Our proposal builds on the facts that (1) components it the fundamental unit of reuse, and (2) components can have introspective interfaces that allow storage and retrieval of information of extra-functional properties. Specifically, we will for each component store a representation of the Execution Time Profile (ETP) that are collected during use of that component. Once a component has been used enough, high confidence in the execution time distribution is obtained. Hence, in addition to the traditional software engineering benefits of CBSE, it also facilitates stochastic schedulability analysis.

2 CBSE for Embedded Systems

During the last decade, the PC-/Internet-oriented software engineering community has achieved tremendous progress in component technologies and component oriented software construction. A component technology consists of a component model, an infrastructure, and tools for creating, composing, and analysing components. When using components, the main idea is to reuse code in a predictable way. The components should be assembled as easy as the Lego™ building-blocks, and the systems should be built up without having to care about the source code; maybe even without access to the source code.

Today, it is possible to download components on the fly and have them executed within the context of another program such as a web browser or a word processor. Software developing companies can purchase off-the-shelf components and embed them into their own software products. Technologies like CORBA [17], Java Beans [19], .NET [16], and other component models are used on a daily basis in software development. However, existing component technologies are not applicable to most embedded computer systems, since they do not consider aspects such as safety, timing, and memory consumption, that are crucial for many embedded real-time systems.

On the other hand, today's software for embedded real-time control systems is characteristically monolithic, platform-dependent and often difficult to maintain, upgrade and modify. In elder software systems, hardware access and operating system calls are often mixed with the application source code, leading to messy and elusive code. Upgrading these systems to a new hardware platform or a new operating system is difficult, error prone and expensive. This is especially problematic since companies traditionally develop new systems in an evolutionary way, i.e. new systems are to a large extend based upon previously developed systems. The development tools often lack support for analysing the software with respect to extra-functional properties, such as timing behaviour and memory consumption. This approach, in trying to adopt existing software to be fit for new environments and new requirements, is far from optimal, and a change of generations in the software development process for embedded systems is approaching.

One major reason to introduce CBSE in embedded real-time systems development is that the software systems are getting more and more advanced. In order to be able to support all the functionality that the users of embedded system demand, novel techniques to master software complexity is needed. Also, for software developing companies to stay competitive their systems must be easy to maintain, upgrade, and modify and the time-to-market has to be kept to a minimum. To be able to tackle these problems, many embedded real-time system developers are interested in using a component based approach.

3 Stochastic Real-Time Analysis

Traditional real-time analysis is based on worst-case assumptions. The first schedulability analysis methods [14] as well as the response-time analysis [9] typically assumes worst-case task-interference and execution times. Also, over the years, efforts to calculate the execution times has focused on calculation of the WCET [18, 7]. However, this is a non-trivial problem, as program timing is difficult to analyse both due to complex program and data flows, and due to non deterministic execution times of program instructions. In fact, no industrial strength tools for WCET-analysis are available on the market.

As real-time systems gets more and more complex, traditional deterministic real-time analysis both becomes more difficult and produce overly pessimistic results. Hence, there is an apparent need for stochastic analysis methods. These stochastic analysis methods should use distributions of execution times instead of the traditionally used worst-case value. Also, stochastic task interference-patterns for schedulability analysis should be accounted for. By taking a stochastic point of view for real-time analysis, less pessimistic analysis results can be provided for the system designers.

3.1 Stochastic Schedulability Analysis

Several stochastic analysis methods have been presented in the real-time research community over the years. Basically these stochastic schedulability analysis methods can be divided into two major groups, where the first group of stochastic analysis methods is based on simplifying, and often unrealistic, assumptions and the second group is using special schedulers that simplify the analysis.

To be able to cope with the complexity of stochastic analysis, virtually all presented research results are based on some restrictive assumptions. One of the more common assumptions is the assumption of the *critical instant*, where all tasks or messages are released at the same time, causing the highest load as well as the worst-case response times. Tia *et al.* [20] present Probabilistic Time Demand Analysis (PTDA) as an extension of the Time Demand Analysis by Lehoczky *et al.* [10]. PTDA is restricted to systems that are using fixed priorities. Also, another requirement is that deadlines have to be less or equal to their corresponding task or message periods. Gardner *et al.* [8] present Stochastic Time Demand Analysis (STDA) as an extension of General Time Demand Analysis by Lehoczky [11]. STDA is better than PTDA in the sense that it can cope with general deadlines. Both PTDA and STDA can use arbitrary execution time distributions.

Other stochastic analysis methods are, e.g., the one presented by Manolache [15] for uniprocessor systems and the one presented by Leulseged *et al.* [13] for multiprocessor systems. However, both of these methods assume that the deadlines of the tasks or the messages are smaller or equal to their corresponding periods. Furthermore, they assume that jobs (tasks or messages) are dropped if their deadlines are to be violated, which is undesirable for many real systems.

Another group of stochastic analysis methods are the Real-Time Queuing theory by Lehoczky [12]. Real-Time Queuing theory can provide stochastic guarantees. However, it requires high traffic load, thus not suitable for a general system configuration.

Finally, several results have been presented requiring a specific scheduler. By using a special scheduling algorithm, tasks can be analysed independently of other tasks in the system, thus simplifying the analysis. Examples of these analysis methods are the one by Abeni [1] for reservation-based systems, and the Statistical Rate Monotonic Scheduling by Atlas *et al.* [2].

Recent work tries to remove most of the limitations of the methods described above. In order to remove the simplifying assumption of critical instant, Diaz *et al.* [5] have presented an extension to PTDA. Their method is based on the usage of Markov processes. The presented method provides exact response-time distributions for any given priority driven scheduler.

3.2 Stochastic Execution-Time Analysis

A problem with stochastic response-time analysis methods and schedulability tests is to find representative distributions of execution times to use in the analysis. Usually the execution times are assumed to follow some known distribution controlled by a few parameters. Whether these distributions are representative or not is normally not considered.

However, several researchers have proposed methods for stochastic estimation of task execution times. One method is proposed by Edgar *et al.* [6]: They show that extreme-value statistical analysis on end-to-end measurements of a tasks, can be used in order to reason about the probability of a violation of the worst case observed response time during testing. Another work is presented by Bernat *et al.* [3], based on the usage of Execution Profiles (EP) to represent execution times of sections of a program. These EPs can then be combined in order to represent the execution times of a whole program. However, issues regarding whether EPs are independent or not are considered.

What we propose in this paper is that the components them self, collect and store information that can be used as representative execution times. In the following section we will present a framework for our approach.

4 Using Components to Facilitate Stochastic Timing Analysis

In order to provide components with representative values for their execution times, we propose that the components themselves log their execution times. By saving information such as: a summary of execution times, all execution times, or some mean values, this information could be collected from the components running in several different applications. This information is then processed, giving a representative description of the components Execution Time Profile (ETP). One of the motivations for using components in a system is that the components can be reused. Hence, by continuously adding information to an ETP, commonly used components will fast get highly representative information describing their execution times.

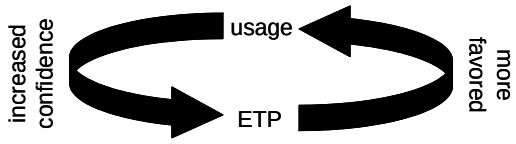

Figure 1. Evolution of Execution Time Profiles

As the components have ETPs of high confidence, the components will become more attractive and used more often. This, in turn, will make the ETPs even more representative as more information on their execution times will be collected. This positive spiral is depicted in Figure 1.

The exact contents and representation of the ETPs could vary from situation to situation. The amount and granularity of the information in the ETPs should be adapted to the desired quality of the analysis to be performed. If only crude estimates of the system timing are needed, the ETPs

need only to store some basic statistical metrics, such as average execution time, standard deviation, and maybe, the 95%-quartile for the execution times. Where as, when high confidence and detailed timing predictions is needed, the ETPs should contain more elaborate data. For instance, histograms of execution times and histograms of blocking times should be of interest.

We have not yet decided which levels of details to support in our future work. However, the examples listed above will be used as initial candidates.

5 Summary and Future Work

In this work-in-progress paper we have presented our ideas regarding the usage of components as a suitable vehicle to extract and store Execution-Time Profiles (ETPs) that can be used for stochastic real-time schedulability analysis.

We have looked into related work already done in the research community regarding stochastic execution-time calculations. Based on this State-Of-The-Art survey (SOTA), we are proceeding towards extending a component model so ETPs easily can be logged and collected in order to continuously improve the confidence of the ETP associated to each component. Using the information from running components on various platforms in various applications, realistic ETPs are collected and stored within the component.

We have in this paper also summarised the current SOTA regarding stochastic real-time analysis methods. Based on this SOTA, we are designing a tool that can be used for stochastic real-time analysis of component based embedded real-time systems. Using this tool, real-time engineers will be able to make cost-reliability tradeoffs by dimensioning hardware resources to achieve the desired reliability.

References

[1] L. Abeni and G. Buttazzo. Stochastic Rate Monotonic Scheduling. In *Proceedings of the* 9^{th} *International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS'01)*, April 2001.

[2] A. Atlas and A. Bestavros. Statistical Rate Monotonic Scheduling. In *Proceedings of the* 19^{th} *IEEE Real-Time Systems Symposium (RTSS'98)*, pages 123–132, Madrid, Spain, December 1998. IEEE Computer Society.

[3] G. Bernat, A. Colin, and S. M. Petters. WCET Analysis of Probabilistic Hard Real-Time Systems. In *Proceedings of the* 23^{rd} *IEEE Real-Time Systems Symposium (RTSS'02)*, pages 289–300, Austin, Texas, USA, December 2002. IEEE Computer Society.

[4] I. Crnkovic and M. Larsson, editors. *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 (RTSS'02)*, pages 289–300, Austin, Texas, USA, December 2002. IEEE Computer Society.

[6] S. Edgar and A. Burns. Statistical Analysis of WCET for Scheduling. In *Proceedings of the* 22^{nd} *IEEE Real-Time Systems Symposium (RTSS'01)*, London, England, December 2001. IEEE Computer Society.

[7] J. Engblom. *Processor Pipelines and Static Worst-Case Execution Time Analysis*. PhD thesis, Uppsala University, Dept. of Information Technology, Box 337, Uppsala, Sweden, April 2002.

[8] 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.

[9] M. Joseph and P. Pandya. Finding Response Times in a Real-Time System. *The Computer Journal - British Computer Society*, 29(5):390–395, October 1986.

[10] J. Lehoczky, L. Sha, and Y. Ding. The Rate Monotonic Scheduling Algorithm - Exact Characterization and Average Case Behaviour. In *Proceedings of* 10^{th} *IEEE Real-Time Systems Symposium (RTSS'89)*, pages 166–171, December 1989.

[11] J. P. Lehoczky. Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines. In *Proceedings of the* 11^{th} *IEEE Real-Time Systems Symposium (RTSS'90)*, pages 201–209, Lake Buena Vista, Florida, USA, December 1990. IEEE Computer Society.

[12] J. P. Lehoczky. Real-Time Queuing Theory. In *Proceedings of* 17^{th} *IEEE Real-Time Systems Symposium (RTSS'96)*, pages 186–195, Los Alamitos, CA, USA, December 1996. IEEE Computer Society.

[13] A. Leulseged and N. Nissanke. Probabilistic Analysis of Multi-processor Scheduling of Tasks with Uncertain Parameter. In *Proceedings of the* 9^{th} *International Conference on Real-Time and Embedded Computing Systems and Applications*, February 2003.

[14] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *Journal of the ACM*, 20(1):40–61, 1973.

[15] S. Manolache. Schedulability Analysis of Real-Time Systems with Stochastic Task Execution Times. Licentiate Thesis No. 985, Dept. of Computer and Information Science, IDA, Linköping University, Sweden, December 2002.

[16] Microsoft. .NET Home Page. http://www.microsoft.com/-net/.

[17] OMG. CORBA Home Page. http://www.omg.org/corba/.

[18] P. Puschner and A. Burns. A Review of Worst-Case Execution-Time Analysis. *Real-Time Systems*, 18(2/3):115–128, May 2000.

[19] SUN Microsystems. Enterprise Javabeans Technology. http://java.sun.com/products/ejb/.

[20] 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.