Introducing a Plug-In Framework for Real-Time Analysis in Rubus-ICE

Kaj Hänninen[1,2], Jukka Mäki-Turja[1], Staffan Sandberg[2], John Lundbäck[2],
Mats Lindberg[2], Mikael Nolin[1,3], Kurt-Lennart Lundbäck[2]
[1]Mälardalen Real-Time Research Centre (MRTC), Västerås, Sweden
[2]Arcticus Systems, Järfälla, Sweden
[3]CC Systems, Uppsala, Sweden
kaj.hanninen@mdh.se

Abstract

In this paper, we present the development of a plug-in framework for integration of real-time analysis methods in the Rubus Integrated Component Environment (Rubus-ICE). We also present the implementation, and evaluate the integration, of two state of the art analysis techniques (i) response-time analysis for tasks with offsets and (ii) shared stack analysis, as plug-ins, in the Rubus-ICE framework.

The paper shows that the proposed framework is well suited for integration of complex analysis methods. However, experience also show that analysis methods are not easily transferred from an academic environment to industry. The main reason for this, we believe, originates from differences in requirements and assumptions between industry and academia.

1. Introduction

Throughout the years, research on analysis and scheduling has been a significant area within the real-time community. A large number of analysis techniques have been proposed for verification of real-time properties in real-time systems (see for example [2, 4, 6, 11, 12, 14, 18–25]). However, many of these techniques are state of the art and non-trivial to understand and even more complex to integrate in an industrial development context. In industrial development, a number of tools are used for design, implementation, analysis and verification. These tools are often manufactured by different vendors. The challenge then becomes to integrate state of the art analysis techniques in an existing tool-suite with tools from different vendors. These difficulties are often hard to overcome; hence many useful analysis techniques never find their way to practical use.

In recent years, plug-in based tools, e.g., Eclipse [5] and JDeveloper [10] etc. have gained popularity. The plug-in concept has several properties that eases the integration of research results in a development environment, for example, (i) allowing the extension of the functionality of a host application by add-on applications (ii) allowing development of add-on plug-ins in isolation, meaning that developers do not need to compile the source code of the plug in with the source code of its host application.

In this paper, we describe the development of a plug-in framework for integration of real-time analysis methods in the Rubus Integrated Component Environment (Rubus-ICE) [1]. We present the Rubus-ICE environment, an IDE targeted for component based development of real-time systems. We then describe the implementation of two novel analysis methods as plug-ins and highlight issues in integration of the plug-ins in a case study.

The contributions of this paper include a proof of concept implementation where state-of-the-art academic results can be deployed in an existing commercial industrial environment. We also report on experiences from transferring academic result to industrial environments and the issues that needs to be dealt with in order to successfully achieve such transfer.

Paper outline. The reminder of this paper is organised as follows. In Section 2 we present the Rubus development environment. Section 3 presents the development of the plug-in framework for Rubus-ICE. In Section 4 we presents the development of two novel analysis methods as plug-ins, for integration in Rubus-ICE. Section Section 5 presents a brief case study on the integration of the analysis plug-ins in the proposed framework. Section Section 6 summarises our experiences in developing the framework and introducing novel research results for industrial use. In Section 7 we conclude the work and outline some future work.

2. Rubus

Rubus is a collection of tools for development of embedded real-time systems. Rubus was introduced for industrial use in 1996. Throughout the years, Rubus has been used by a number of companies, .e.g., [3, 7, 13, 15, 26] for development of safety critical as well as less critical vehicular software. Although successfully used by real-time developers, the tools in the Rubus framework have by tradition been tightly coupled with each other, making it difficult to integrate additional analysis methods in the framework. Over the years, the tools-suite has evolved to support novel requirements in development. The current version of the tool-suite, Rubus-ICE, is aimed to be plug-in based to facilitate integration of third party applications, such as real-time analysis methods, in the framework.

2.1. Rubus-ICE

Rubus-ICE is an IDE consisting of set of tools for systems engineering, design and analysis of component-based real-time systems. The four core elements of Rubus-ICE are as follows:

- Designer: A graphical design tool for component based modelling of systems.
- Compiler: A tool that verifies syntax of the model data created with the designer.
- Builder: A tool that passes design-data in sequence to any number of user specified plug-in modules.
- Coder: A tool that generates the RTOS specific requirements defined by the user.

To exemplify the steps involved in using Rubus-ICE, assume that a developer initially creates a component-based design using Designer. The Designer saves the design in XML format. The compiler then parses the design representation and verifies the syntax of the design. The compiler creates an intermediate representation (ICCM file) of the design. The ICCM file is then used by the builder. Figure 1 shows the sequence in which the core elements of Rubus-ICE are executed.

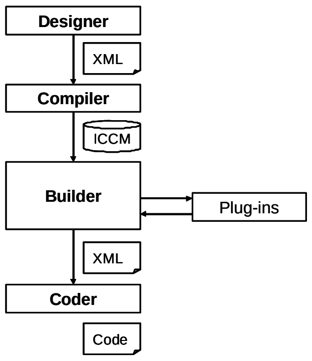

Figure 1. Rubus-ICE with a plug-in framework

3. Plug-in framework for Rubus-ICE

In this section we describe the development of a plug-in framework for Rubus-ICE. The aim of the framework is to enable integration of novel real-time analysis methods, as plug-ins, in the IDE. Facilitating integration of third-party developed analysis methods is of specific interest. Hence, the framework should allow a plug-in developer to implement plug-ins in isolation, i.e., without having the Rubus tool suite at hand, and deliver the plug-ins as binaries or source code. In the following, we describe the overall requirements on the plug-in framework. We then outline the requirements that need to be fulfilled for plug-ins to be included in the framework. We also present the development of an application programming interface (API) for development of Rubus-ICE plug-ins.

3.1. Requirements on the plug-in framework

We start by stating the high level requirements on the plug-in framework. From a developer and a user perspective the following requirements should be fulfilled by the framework:

- A plug-in should be developed as stand alone applications performing a specific task.
- A user should be able to choose, by configuring a build, to execute any available plug-in during the build. Hence, the plug-ins should interface the Builder tool in Rubus-ICE.
- Plug-ins should execute in sequence, meaning that a plug-in should execute to completion and terminate before the next plug-in is executed.
- A user should be able view the progress of a plug-in and to abort the execution of a plug-in if needed.

Figure 1 shows the sequence in which the core elements of Rubus-ICE, including plug-ins, are to be executed. The requirements on the framework are motivated by the following. For example, a user might be interested in analysing only temporal aspects of a design, during certain phases in development. Later on, the user might be interested in analysing both temporal and spatial aspects. Hence it should be possible to enable and disable the execution of plug-ins between the builds. The requirement that each plug-in should perform one specific task is required to prevent several plug-ins to perform similar, possibly time consuming, analysis during build. For example, if several analysis algorithms require response times to be calculated as input to the algorithms, then the response-time analysis should be developed as a separate plug-in that is executed only once. The fact that plug-ins are required to execute in sequence facilitate the possibility of several plug-ins to collaborate and solve a larger task. Moreover, in the research community, several analysis methods are proposed as extensions of previously published methods. Requiring the plug-ins to execute in a sequence eases the integration of the extensions in the IDE. User interaction and the possibility to abort the execution of a plug-in is motivated by the fact that the timing complexity of an analysis method may increase dramatically if a system is changed between two builds. For example, analysis methods with exponential complexity may actually perform well, i.e., deliver results in reasonably amount of time, for a certain system. However, for such algorithms, even small changes in system parameters may dramatically increase the time to obtain results. In these type of situations, the analysis may be unusable and aborted by the user. In addition, providing feedback to users when a plug-in fails or when the results of a plug-in differs from what is expected, is important.

3.2. Requirements on Rubus-ICE plug-ins

Many analysis methods developed in a research context assume that certain assumptions are fulfilled for the analysis to be valid. For example, in the research community, an analysis algorithm is generally developed for a specific system model, i.e., the results of the analysis is only valid if the correct system model is used. Thus, for each plug-in, the supported system model, i.e., properties and attributes of the supported system, must be specified.

To simplify verification of plug-ins, we require plug-ins to adhere to the following execution sequence: (i) reading required system attributes, (ii) executing the functionality of the plug in and (iii) writing results to the ICCM file. Hence, each plug-in should have a required interface, an internal behaviour and a provided interface. Accessing system attributes should be done by service request provided by an Application Programming Interface.

Each plug-in must have its error handling routines specified. This includes specifying (i) the type of error that the plug-in handles, and (ii) how these error are handled. In addition, in an industrial context, interaction with the user is imperative. Hence, each plug-in needs to define an interface against the user. This interface should provide, e.g., information of the progress of the plug-ins.

The fact that a user should be able to choose, by configuring a build, to load and execute any available plug-in during the build implies that plug-in can be delivered as Dynamic Link Libraries (DLLs) or as source code (C or C++ code).

In essence, for each Rubus-ICE plug-in, the following should specified: (i) the system model supported by the plug-in, (ii) the type of data required by the plug-in, (iii) the type of data produced by the plug-in, (iv) the type of errors handled by the plug-in and (v) user interface.

3.3. Defining an API for plug-ins

To support implementation of plug-ins, we defined an API (Application Programming Interface). The API specifies and provides a uniform way to access services that may be needed by plug-ins. Currently, the API supports common services for the system model defined by the Rubus Component Model (RubusCM) [9]. In defining the API, we considered common assumptions made by researcher developing analysis algorithms. We also considered common attributes and properties that need to be available for analysis algorithms. For instance, the API provide possibilities to extract transactions, tasks, task attributes, task dependencies, execution policies, execution schedules, memory-models and so on, from the design. In addition, the API provide services to append the results, produced by a plug-in, in the ICCM file. These results may then be used by subsequent plug-ins.

4. Developing analysis plug-ins

In this section, we describe the development and integration of two analysis plug-ins for Rubus-ICE. The plug-ins are intended to be stand alone applications computing: (i) the worst case response-time (RTA) [16] of tasks and (ii) the maximum shared stack usage (SSA) [8] of the system. Both the RTA and SSA algorithms have originally been implemented for research purposes, e.g., for evaluating the efficiency of the RTA and SSA algorithms. In a research context, the algorithms have been part of a larger application consisting of a task generator, a package for statistics an a graph generator (see Fig. 2).

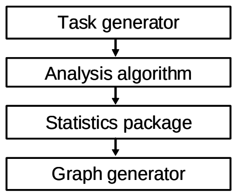

Figure 2. Analysis method in a research context

As preparation, the functionality belonging to the algorithms were extracted from the application. These functions constitute the core of the RTA and the core of the SSA plug-ins. According to the requirements specified in Section 3.2, the system information is accessed by the provided API. Fig. 3 shows a conceptual structure of the RTA and SSA plug-ins.

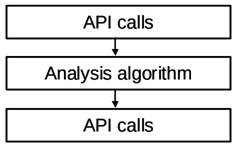

Figure 3. Analysis method in a plug-in context

The RTA plug-in is based on work by Mäki-Turja and Nolin [16]. The plug-in will compute the worst case response-time of tasks. The SSA plug-in is based on work by Hänninen *et al.* [8] and will compute an upper bound on shared system stack. We start by specifying the system model for the RTA and SSA plug-ins. We then define the required and provided interfaces as well as the error handling and user interface of each plug-in.

The system model in both [16] and [8] is an offset based model with transactions (a transactions is defined by a set of tasks with timing dependencies). Each transaction consist of one or more tasks. Tasks, in turn, have common real-time attributes such as worst case execution times, deadlines, priorities etc. For the RTA and SSA plug-ins, this implies that we need to extract transactions and task attributes from the design, execute the analysis and store the analysis results. Furthermore, from [16] we know that the RTA interface should support the following:

- The RTA plug-in require, (i) Transactions with specified Period time (or MINT), (ii) the WCET, Offset and the Priority of the tasks in each transaction.
- The RTA provides the worst case response time (WCRT), relative to the activation of the transaction, of each task.

From [8] we know that the SSA interface should support the following:

- The SSA plug-in require, (i) Transactions participating in stack sharing, (ii) the WCRT, Offset, Stack usage and the Priority of the tasks in each transaction.
- The SSA provides an upper bound on shared stack usage of the transactions.

Recall that since plug-ins are executed in succession in Rubus-ICE, each plug in must specify the data it needs and the data it produces. This is required for correct execution sequence of the plug-ins. For example, the above shows that the SSA plug in require worst case response-times for stack analysis, i.e., the RTA plug-in must be executed before the SSA plug-in, showing that analysis methods may have intricate dependencies that need to be considered when establishing the execution sequence of plug-ins.

When designing the error handling of the plug-ins, we observed that if something fails during analysis, the plug-in must be able to handle and isolate the fault. The plug-in must also inform the host application of the fault. This is needed to isolate, i.e., to prevent the fault or erroneous values to propagate. For example, if the system is overloaded, i.e., the processor utilisation exceeds 100%, the response time analysis, being an iterative method, may never terminate. An even worse scenarios could occur if a variable overflows, then the RTA might actually terminate producing erroneous results. In a controlled research setting, this might not be a problem, since task generators are often configured to produce schedulable task sets as input to an analysis method. In an industrial setting, this assumption no longer hold. It is obvious that conditions such as overloads, variable overflows etc. need to be handled and dealt with properly. We also defined the actions that should be taken if, during analysis, the response-time of a task is larger than its deadline, i.e., the task is missing its deadline. The question then is, should we continue or abort the analysis. Although, this kind of situation might not be considered as an error, however, it might affect the execution of subsequent plug-ins. For instance, the algorithm in the RTA plug-in put no restrictions on response-times, i.e., response-time may be larger than the deadlines without affecting the correctness of the analysis. The SSA algorithm, on the other hand, require that a response-time is smaller (or equal) than the deadline.

The following list the error handling that need to be supported by the RTA and SSA plug-ins:

- The values of the read task attributes need to be checked.
- Overload conditions need to be checked during analysis.
- Variable overflow need to be checked during analysis.

For both the RTA and SSA plug-in, we define a simple interface against the user. The interface provides information of the progress of the plug-in and a summary of the analysis results. Fig. 4 shows the complete structure of the RTA and SSA plug-ins.

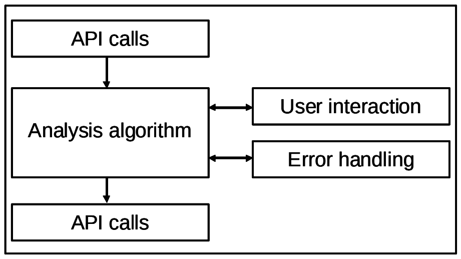

Figure 4. Analysis plug-in in an industrial context

5. Adding plug-ins to Rubus-ICE - A case study

The plug-in framework, described in Section 3, allows users of Rubus-ICE to include third part developed plug-ins to the Rubus-ICE framework. To point out issues that may occur when adding plug-ins to the framework, we integrated the RTA and SSA plug-ins in Rubus-ICE. We believe that the RTA and SSA plug-ins represents typical analysis proposed by researchers and that the integration of RTA and SSA plug-ins relieve issues that may be encountered when integrating other type of real-time analysis methods in Rubus-ICE.

The prerequisites for the integration was as follows:

- The plug-ins were implemented, adhering to the requirements in Section 3.2, and delivered as source code to the integrator.
- Both the RTA and SSA plug-ins should be integrated in Rubus-ICE.

The plug-ins were integrated in Rubus-ICE by a developer at Arcticus Systems. The developer had no previous experience of integrating real-time analysis methods, but was familiar with the overall objective of the plug-in framework and had been involved in specifying the requirements on the framework (Section 3.1). During integration, the developer was asked to note all issues that occurred during the integration. Integrating the plug-ins include (i) enabling configuration of a build sequence, (ii) establishing a correct execution sequence of plug-ins, (iii) verifying the functionality of the plug-ins and (iv) verifying the error handling of each plug-in. Even though the functionality, including error handling, of the RTA and SSA plug-ins have been verified in isolation, the integration in-itself may introduce unexpected errors, hence the plug-ins need to be verified once integrated. The following summarises the experiences of the integration as encountered by the integrator:

- Integration of the RTA and SSA plug-ins in Builder, i.e., enabling configuration of the build sequence to include the RTA ans SSA plug-ins, proceeded without notable difficulties.
- Establishing the execution sequence of the plug-ins was eased by the interface specifications included with the plug-ins.
- Verification of the functionality of each plug-in was experienced as troublesome. The integrator needed to consult the creators of the plug-ins to perform the verification. To verify the functionality of, for example, the RTA plug-in, example systems with only a few tasks were created. The results of the plug-in then needed to be verified by hand. It is obvious that larger systems are intractable to analyse by hand. Hence only small systems, with varying architecture, were possible to analyse.
- Certain type of error handling was difficult to verify. For example, verifying that variable overflows was handled properly was difficult, simply because it was difficult to create systems, or modify the attributes of the system, in such way that it resulted in variable overflows at the same time as the results of the plug-in seemed valid (see error handling in Section 4).
- When verifying the functionality of the RTA plug-in, the integrator discovered that the RTA plug-in sometimes produced pessimistic, albeit safe, response-times. When investigating the reason to this, we discovered that the API service extracting transactions from the design needed to be modified. The extraction, although being correct in a sense, failed to exploit the benefits of the analysis. Specifically, since the analysis is developed for an offset based system (offsets represents timing dependencies), these dependencies must be extracted from the design, and the better the extraction can represent the timing dependencies, the tighter the results from the analysis.

Altogether, the integration of the RTA and SSA plug-in in Rubus-ICE was experienced as fairly easy. We believe that the fact that both plug-ins were developed according to the requirements outlined in Section 3.2, facilitated the integration. Verification of the functionality was experienced as the most difficult task and the most time consuming activity. A continuous communication between the integrator and the plug-in developer was needed during the integration. This clearly demonstrates that the work of a plug-in developer do not end with the delivery of a plug-in. Even though the plug-in concept allows integration of third-party developed software, such as novel analysis methods, in a larger context, we believe that when transferring research results for industrial use, especially for use in development of safety critical systems, collaboration between the integrator and the plug-in developer is needed throughout the whole process for a successful end result.

6. Experiences summarised

We initiated this work with the aim of developing a plug-in framework to enable integration of third-party developed software. We showed, by integrating two plug-ins in the framework, that it is possible to add complex real-time analysis methods to Rubus-ICE. However, we discovered that a considerable amount of work is needed to prepare and integrate research results for industrial use. The main reason for this, we believe, is that the requirements and assumptions on analysis methods, in an industrial context, differs from the requirements/assumptions in a research context. For example, in an industrial context many analysis methods are used in developing safety critical software, implying that stringent error handling as well as thorough verification of the functionality is needed before analysis methods are accepted for industrial use. We also noticed that verification of the plug-ins, after being integrated in Rubus-ICE, required the help of the plug-in developer, since analysis methods are often very hard to understand and too difficult to verify by non-experts. When defining the user interface of the plug-ins, we discovered that it was non-trivial to provide constructive feedback to the users (compare to finding 9 in [17]). For example, since the RTA plug-in can be used to verify the schedulability of a system, the plug-in may occasionally discover that a system is unschedulable. The question then is, should we suggest modifications (e.g. changing priorities) of the attributes in the system, i.e., guiding the developer to possibly end up with a schedulable system. In many cases there are a large number of possible reasons why a system is unschedulable, hence suggesting constructive modifications is non-trivial.

Although we managed to integrate two novel analysis methods (the RTA and SSA plug-ins) in the framework, several issues still remain to addressed, for example:

- How can we guarantee that a plug-in only does what it is meant to do. For example, in the current framework it is possible for a plug-in to overwrite any values in the ICCM. These value may later be used by subsequent plug-ins and result in erroneous results.

- How should the output results from a plug-in be named in the ICCM. For example, if two plug-ins collaborate to solve a larger task, i.e., a plug-in reads the results of the other plug-in, then the plug-in reading the values produced by another plug-in need to have knowledge of the naming, simply to be able to fetch the values from the ICCM. Currently, there is no standard notation within the real-time community on how to name the results produced by an analysis.

- How should we aggregate several plug-ins into a single assembly. Consider, for instance, priority assignment (assigning the dispatching priority to tasks). In real-time engineering, the arduous work of priority assignment is sometimes performed by hand. With the assigned priorities at hand, response time analysis can then be performed to verify schedulability of a system. In case the system turns out to be unschedulable, the priorities are modified and response-time analysis is performed again. The process is basically an iteration of priority assignment and schedulability check by response-time analysis. The iteration is often performed until a schedulable system is found. This could be an automated procedure using two plug ins in the proposed framework, one that assign priorities and one that performs response-time analysis (e.g., the RTA plug-in). However, iterating the execution of these plug-ins, require them to be grouped in a single plug-in. Currently, there is no other way of iterating the execution of two or more plug-ins in the framework.

7. Conclusions and future work

In recent years, plug-in based development tools has increased in popularity. The plug-in concept enable third-party developed software to extend the functionality of a host application by add-ons, giving tool manufacturers a considerable simple way of adding new features and enhancing the value of their products. Developers can also benefit from the plug-in concept in the sense that plug-ins can be developed in isolation and do not require deep knowledge of the host application utilising the plug-in. These facts make the plug-in concept especially interesting when transferring complex research result for industrial use. Most of the published results from the research community are far too difficult to understand and even harder to implement for laymen, implying that many useful results never find their way to practical use.

In this paper we presented the development of a plug-in framework for Rubus Integrated Component Environment, an IDE for development of embedded real-time systems. The plug-in framework aims at facilitating integration of novel analysis methods in the framework. We presented the framework and implemented two novel analysis methods as plug-ins. The plug-ins were integrated in the framework by a developer without previous expertise in analysis methods for real-time systems. We showed that the framework is well suited for integration of complex analysis methods, however, we also showed that a considerable amount of modifications of analysis methods are needed to adapt them for industrial use. In addition, a continuous communication between the researchers developing the plug-ins and the industrial representative integrating the plug-ins, was needed throughout the process.

As future work, we plan to extend the application programming interface to support other type of system models. Currently the Rubus system model is the only one supported. We also intend to define a way to aggregate several plug-ins into a larger one (an assembly consisting of several separate plug-ins). We believe that many engineering activities, such as priority assignment, schedule creation and task allocation etc. could be automated by aggregated plug-ins.

References

[1] Arcticus systems. Web page, http://www.arcticus-systems.se.

[2] N. Audsley, A. Burns, K. Tindell, M. Richardson, and A. Wellings. Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling. *Software Engineering Journal*, 8(5):284–292, 1993.

[3] BAE Systems Hägglunds. Web page, http://www.-baesystems.com/hagglunds.

[4] A. Burns, K. Tindell, and A. Wellings. Effective Analysis for Engineering Real-Time Fixed Priority Schedulers. *IEEE Transactions on Software Engineering*, 22(5):475–480, May 1995.

[5] Eclipse - an open development platform. Web page, http://www.eclipse.org.

[6] A. Ermedahl, H. Hansson, and M. Sjödin. Response-Time Guarantees in ATM Networks. pages 274–284. IEEE Computer Society Press, December 1997. URL: http://-www.docs.uu.se/~mic/papers.html.

[7] Haldex traction systems. Web page, http://www.haldex-traction.com/.

[8] K. Hänninen, J. Mäki-Turja, M. Bohlin, J. Carlsson, and M. Nolin. Determining maximum stack usage in preemptive shared stack systems. In *Proceedings of the 27th IEEE Real-Time Systems Symposium*, Dec 2006.

[9] K. Hänninen, J. Mäki-Turja, M. Nolin, M. Lindberg, J. Lundbäck, and K.-L. Lundbäck. Supporting engineering requirements in the rubus component model. Technical report.

[10] Oracle JDeveloper Overview, An Oracle White Paper. Web page, http://www.oracle.com/technology/products/-jdev/collateral/papers/1013/jdev1013_overview.pdf.

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

[12] D. Katcher, H. Arakawa, and J. Strosnider. Engineering and analysis of fixed priority schedulers. *IEEE Transactions on Software Engineering*, 19(9):920–934, September 1993.

[13] Knorr-bremse. Web page, http://www.knorr-bremse.com.

[14] J. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. pages 201–212, December 1990.

[15] Mecel. Web page, http://www.mecel.se/.

[16] J. Mäki-Turja and M. Nolin. Tighter response-times for tasks with offsets. In *Real-time and Embedded Computing Systems and Applications Conference*, Göteborg, Sweden, August 2004. Springer-Verlag.

[17] C. Norström, M. Gustafsson, K. Sandström, J. Mäki-Turja, and N. E. Bånkestad. Experiences from Introducing State-of-the-art Real-Time Techniques in the Automotive Industry. In *Eigth IEEE International Conference and Workshop on the Engineering of Computer-Based Systems*. IEEE Computer Society, April 2001.

[18] S. Punnekkat. *Schedulability Analysis for Fault Tolerant Real-time Systems*. PhD thesis, University of York, June 1997.

[19] L. Sha, T. Abdelzaher, K.-E. Årzén, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. K. Mok. Real Time Scheduling Theory: A Historical Perspective. *Real-Time Systems*, 28(2/3):101–155, 2004.

[20] L. Sha, R. Rajkumar, and J. Lehoczky. Task scheduling in distributed real-time systems. In *IEEE Industrial Electronics Conference*, 1987.

[21] L. Sha, R. Rajkumar, and J. Lehoczky. Priority Inheritance Protocols: an Approach to Real Time Synchronization . *IEEE Transactions on Computers*, 39(9):1175–1185, September 1990.

[22] K. Tindell. An extendible approach for analyzing fixed priority hard real-time tasks. Technical Report YCS189, Dept. of Computer Science, University of York, England, 1992.

[23] K. Tindell and A. Burns. Fixed Priority Scheduling of Hard Real-Time Multimedia Disk Traffic. *The Computer Journal*, 37(8):691–697, 1994.

[24] K. Tindell and J. Clark. Holistic Schedulability Analysis For Distributed Hard Real-Time Systems. Technical Report YCS197, Real-Time Systems Research Group, Department of Computer Science, University of York, November 1994. URL ftp://ftp.cs.york.ac.uk/pub/-realtime/papers/YCS197.ps.Z.

[25] K. Tindell, H. Hansson, and A. Wellings. Analysing Real-Time Communications: Controller Area Network (CAN). pages 259–263. IEEE, IEEE Computer Society Press, December 1994.

[26] Volvo Construction Equipment. Web page, http://www.volvoce.com.