Mälardalen University Licentiate Thesis
No.47

Transformation of component models to real-time models

Optimizing resource usage

Johan Fredriksson

April 2005

Department of Computer Science and Electronics
Mälardalen University
Västerås, Sweden

ISSN 1651-9256
ISBN 91-88834-55-7
Printed by Arkitektkopia, Västerås, Sweden
Distribution: Mälardalen University Press

Abstract

Industry is constantly looking for new developments in software for use in increasingly complex computer applications. Today, the development of component-based systems is an attractive area for both Industry and Academia. The systems we focus on in this thesis are embedded computers, in particular those in automotive systems. A modern car incorporates several embedded computers that control different functions of the car, e.g., anti-spin and anti-lock breaks.

The main purpose of this thesis is to investigate how component technologies for use in embedded systems can reduce resource usage without compromising non-functional requirements, such as timeliness.

The component-technologies available have not yet been used extensively in the vehicular domain. To understand why this is the case we have conducted a survey and performed evaluations of the requirements of the vehicular industry with respect to software and software development. The purpose of the evaluation was to provide a foundation for defining models, methods and tools for component-based software engineering.

The main contribution of this work is the implementation and evaluation of a framework for resource-efficient mappings between component-models and real-time systems. Few component technologies today consider the mapping between components and run-time tasks. We show how effective mappings can reduce memory usage and CPU-overhead. The implemented framework utilizes genetic algorithms to find feasible, resource efficient mappings.

We show how component models designed for resource constrained safety-critical embedded real-time systems can use powerful compile-time techniques to realize the component-based approach and ensure predictable behaviour.

Further, we propose a resource reclaiming strategy for component-based real-time systems, to decrease the impact of pessimistic execution time predictions. In our approach, components run in different quality levels as unused processor time is accumulated.

Det är vackrast när det skymmer.
All den kärlek himlen rymmer
ligger samlad i ett dunkelt ljus
över jorden,
över markens hus.

Allt är ömhet, allt är smekt av händer.
Herren själv utplånar fjärran stränder.
Allt är nära, allt är långt ifrån.
Allt är givet
människan som lån.

Allt är mitt, och allt skall tagas från mig,
inom kort skall allting tagas från mig.
Träden, molnen, marken där jag går.
Jag skall vandra -
ensam, utan spår.

Per Lagerkvist - ¨Det är vackrast när det skymmer¨

-För att livet är för kort att slösas bort.

To Marie, Erika, Inger and Tommy

Without whom life would be a waste!

Preface

When I began my undergraduate studies at Mälardalen University I intended to study for three years, and no more. However, after three years the studies had become so interesting that I decided to continued for a further one and a half years. After this one and a half years I again discovered that my need for more knowledge was unsatisfied. Having been a Ph.D. student for two years I find that the fascination continues and I now wonder where this is going to end?

However, without the blessing, guidance and attention from my supervisors Prof. Ivica Crnkovic and Dr. Kristian Sandström, I would never have reached this far in my research. It is always a pleasure to work, discuss and relax with you guys! I also want to thank Dr. Mikael Nolin, Prof. Hans Hansson, Prof. Christer Norström for your advice and guidance. I also want to express my gratitude to Harriet Ekwall for always solving the practically unsolvable!

My work as a Ph.D. candidate would not have been as fruitful (or as enjoyable) without the discussions with and the cooperation and companionship of my fellow Ph.D. students. My special thanks goes to Mikael Åkerholm and Anders Möller for their cooperation, encouragement and friendship. I want to thank the department of Computer Science and Engineering in general and especially Markus N, Johan A, Rikard, Dag, Thomas N, Jonas, Anders P, Dr. Anders W, Daniel S, Joel, Jukka, Daniel F, Radu, Damir, Larissa, Thomas L for making this time so rewarding. And for all the time given me during my basic education I especially want to thank Christian 'Cribbe' Andersson.

I want to thank my mother, father and sister for all their love, and all my relatives for making my life great.

Finally, I thank thee Marie, for all thy love and encouragement!

Johan Fredriksson
Västerås, April 13, 2005

List of Publications

Publications included in the thesis

Conferences and Workshops

Paper A Johan Fredriksson, Kristian Sandström and Mikael Åkerholm, *Optimizing Resource Usage in Component-Based Real-Time Systems*, In proceedings of the 8th International Symposium on Component-based Software Engineering (CBSE8), St. Louis, US, May 2005.

This paper presents work on how components can be allocated to real-time tasks with respect to schedulability, resource usage and memory consumption. The allocation methods utilize genetic algorithms to find feasible allocations, while optimizing for low resource usage and memory consumption.

Johan has initiated and led the research. He has been involved in all parts of this paper.

Paper B Johan Fredriksson and Mikael Åkerholm, Radu Dobrin, Kristian Sandström *Attaining Flexible Real-Time Systems by Bringing Together Component Technologies and Real-Time Systems Theory*, In Proceedings of the 29th Euromicro Conference, Component Based Software Engineering Track Belek, Turkey, September 2003.

This paper presents work on how run-time mechanisms and analysis can be used to guarantee the timeliness in the system, while utilizing resource reclaiming during run-time to provide different levels of quality.

Johan has been involved in all parts of this work.

Paper C Kristian Sandström, Johan Fredriksson and Mikael Åkerholm, *Introducing a Component Technology for Safety Critical Embedded Real-Time Systems*, In International Symposium on Component-based Software Engineering (CBSE7) Edinburgh, Scotland, May 2004.

This paper presents work on how a component based software engineering can be used for embedded systems with high requirements on predictability, resource efficiency and low memory footprint. The development process is divided into distinct phases of design, compilation and run-time, providing expressive design possibilities, and resource effective run-time models by powerful compile-time techniques.

Johan has been involved in all parts of the work with special focus on the compilation and run-time steps.

Paper D Mikael Åkerholm, Johan Fredriksson, Kristian Sandström and Ivica Crnkovic, *Quality Attribute Support in a Component Technology for Vehicular Software*, In Fourth Conference on Software Engineering Research and Practice in Sweden Linköping, Sweden, October 2004.

This paper is based on a survey where representatives from different vehicular companies have prioritized a number of quality attributes. The paper presents the results of the survey and discusses the impact on component technologies. Johan has participated in the evaluation and discussion parts.

Paper E Anders Möller, Mikael Åkerholm, Johan Fredriksson and Mikael Nolin, *Evaluation of Component Technologies with Respect to Industrial Requirements*, In Euromicro Conference, Component-Based Software Engineering Track Rennes, France, August 2004.

This paper is an evaluation of existing component technologies for the vehicular systems. The evaluation is performed with respect to their suitability for the vehicular domain. The evaluation is based on literature studies and interviews capturing requirements from the industry.

Johan's part of this work has been to provide knowledge about the component technologies evaluated and participate in the evaluation process.

Other publications, not included in the thesis

Conferences and Workshops

- Johan Fredriksson, Mikael Åkerholm and Kristian Sandström, Calculating Resource Trade-offs when Mapping Component Services to Real-Time Tasks, In Fourth Conference on Software Engineering Research and Practice in Sweden Linköping, Sweden, October 2004.

- Anders Möller, Mikael Åkerholm, Johan Fredriksson and Mikael Nolin, Software Component Technologies for Real-Time Systems - An Industrial Perspective, In WiP Session of Real-Time Systems Symposium (RTSS) Cancun, Mexico, December 2003.

- Ivica Crnkovic, Igor Čavrak, Johan Fredriksson, Rikard Land, Mario Žagar and Mikael Åkerholm, On the Teaching of Distributed Software Development, In 25th International Conference Information Technology Interfaces Dubrovnik, Croatia, June 2003.

Technical Reports

- Johan Fredriksson, Achieve consistent mappings between component models and real-time models - Licentiate Thesis Proposal, Technical Report, MRTC, June, 2004

- Mikael Åkerholm, Kristian Sandström and Johan Fredriksson, Interference Control for Integration of Vehicular Software Components, MRTC Report ISSN 1404-3041 ISRN MDH-MRTC-162/2004-1-SE, Mälardalen Real-Time Research Centre, Mälardalen University, May 2004.

- Anders Möller, Mikael Åkerholm, Johan Fredriksson and Mikael Nolin, An Industrial Evaluation of Component Technologies for Embedded-Systems, MRTC Report ISSN 1404-3041 ISRN MDH-MRTC-155/2004-1-SE, Mälardalen Real-Time Research Centre, Mälardalen University, February 2004.

- Mikael Nolin, Johan Fredriksson, Jerker Hammarberg, Joel G Huselius, John Håkansson, Annika Karlsson, Ola Larses, Markus Lindgren, Goran Mustapic, Anders Möller, Thomas Nolte, Jonas Norberg, Dag Nyström, Aleksandra Tesanovic, and Mikael Åkerholm, Component Based Software Engineering for Embedded Systems - A literature survey, MRTC

Report ISSN 1404-3041 ISRN MDH-MRTC-102/2003-1-SE, Mälardalen Real-Time Research Centre, Mälardalen University, June 2003.

- Mikael Åkerholm and Johan Fredriksson, A Sample of Component Technologies for Embedded Systems, Technical Report, November 2004.

Contents

I

Thesis

Chapter 1

Introduction

During recent decades processors have become more powerful and more cost efficient. They are now of interest in areas where not used before, permitting the development of ever more complex system applications. To manage the increasing complexity of applications, Industry is constantly looking for new software development strategies. One paradigm found to be of use in desktop computing systems is Component-Based Software Engineering (CBSE). CBSE has been under development since at least 1968 and is based on the idea of reusing existing components, in much the same way as standard components are used in other engineering disciplines [1, 2, 3].

Processors are used in what is known as embedded systems for control all kinds of devices and technical systems used in society, ranging from mp3-players to nuclear plants. The fastest growing demand for microprocessors is for use in embedded systems and in recent years, over 99.8% [4] of all processors produced are incorporated in embedded systems. IEEE [5] has provided a common definition of embedded systems as follows:

> *A computer system that is a part of a larger system and performs some of the requirements of that system; for example a computer system used in an aircraft or rapid transit system.*

Embedded systems have requirements that are not regarded in desktop applications, such as low memory utilization, low processor overhead and predictability. Many embedded systems are safety-critical because they control applications in our society. If these applications malfunction they can have disastrous consequences. Hence, non-functional characteristics (such as reliability) are very important in these types of applications. Non-functional re-

quirements are attributes indicating in some way the quality of the system. One important class of such requirements are real-time requirements. These requirements define within what time a task must be performed. More specifically in [6] Stankovic states:

> *Real-time systems are computer systems in which the correctness of the system depends not only on the logical correctness of the computations performed, but also on which point in time the results are provided.*

In the real-time and embedded systems domains there are many theories, methods and tools. These methods use a number of real-time properties, such as worst-case execution time (WCET), execution period, deadlines, etc., and terms such as tasks and scheduling, in reasoning about timing and other requirements.

The notion of components is rarely used in real-time systems. On the other hand, current component technologies usually do not include non-functional properties typical of real-time systems. To be able to use a component approach, which makes the system development process more efficient, and at the same time guarantees system behaviour, both how the component technology uses non-functional properties, and how components are allocated to the run-time systems are important.

The purpose of this work is to demonstrate how component models can use non-functional properties to support real-time analysis; and in particular to provide methods for resource-efficient allocation of components to run-time tasks, optimized for, e.g., memory and performance. Further we demonstrate how a light weight component framework can use more advanced features such as multiple versions, with no negative effect on the real-time properties.

1.1 Research Motivation

The research leading to this thesis was motivated by the increasing complexity of modern embedded systems, such as, e.g., advanced engine controls and anti-skid systems. The significance of the research is confirmed by the vehicular industry performing research in the same area, e.g., in projects such as AUTOSAR [7]. Today issues relevant to embedded component-based systems such as real-time and resource efficiency are often addressed outside CBSE. There are many methods and theories for, e.g., real-time analysis, but very few in relation to CBSE. To handle real-time and resource efficiency in CBSE for embedded systems we consider several aspects, including:

- Optimization with respect to speed and memory for component-based systems.
- Support for non-functional properties.
- Prediction of system properties, e.g., timeliness, etc.
- Light-weight component frameworks for more advanced features.

The significance of these issues is confirmed by research referred to in [8], in which the authors have identified needs and priorities for research related to CBSE for embedded systems.

1.2 Embedded Real-Time Systems

We do not need to search far to find an example of an embedded real-time system in a modern everyday appliance. This thesis focuses on vehicular systems, using a modern vehicle as an example. The engine is controlled by a complex real-time system, measuring the airflow to the engine, pumping in just the right amount of fuel and igniting this in each cylinder at the exact right moment. The anti-lock breaks are controlled by a real-time system, continuously monitoring and controlling the breaks to ensure the maximum breaking effect. In the unlikely case of a collision, an embedded real-time system will detect the impact and deploy the airbag at exactly the right point in time. What is common to all these systems is that they are parts of a bigger system and their actions have to be delivered at a specified interval in the time. If they fail to deliver their services at the right time, the consequences can lead to low performance, material damage or in the worst scenario, loss of human life.

Though this thesis focuses on vehicular systems, the research is applicable in a wider range of application domains. Thus, in this section, we briefly describe three application areas of embedded real-time systems, *vehicular systems*, *consumer electronics* and *industrial embedded applications*. Complex embedded systems with requirements on timeliness are used in all these domains, which are characterized by large production volumes and product lines. Other domains in which component-based engineering for embedded systems can be used are, e.g., *medical* and *telecom systems*.

Vehicular systems The complexity of the computerized vehicular systems has increased over the last decade, and component based development is envisioned as a promising future approach to increasing productivity. The

cost reduction impact from, e.g., lowering the memory requirement by enabling efficient mappings between components and tasks, and thereby saving a number of bytes of memory can be substantial. Many parts of a vehicle, e.g., the breaking and the airbag systems have strict requirements with respect to reliability, safety, and predictability. It is therefore vital that the real-time properties are maintained even though resource usage is minimized.

Consumer electronics The consumer electronics domain embraces, among many other products television receivers, DVD-players, cell phones. Many consumer products have real-time properties which must be guaranteed, e.g., a smooth playback, or synchronization of picture and sound. It is often important to maintain timeliness, especially for high-end products, to maintain a competitive edge. The component-based approach has successfully been used for several years by, e.g., Philips [9].

Industrial embedded applications In the domain of industrial applications, such as industrial robots and automation systems, the systems are often more extensive and complex than those in vehicular and consumer electronics. A single node in an industrial application can consist of hundreds or thousands of software modules. Industrial applications often have strict requirements on timing and reliability, safety and performance, but must also be flexible, portable and scalable. By improving performance and reducing the memory requirement, it may be possible to add more sophisticated or more advanced features to an industrial application without adding to its costs which may give a competitive edge.

Before continuing we will provide some basic terminology of Real-Time Systems.

1.2.1 Real-Time Systems Terminology

In *hard real-time systems*, a program delivering a result after its latest acceptable time, i.e., its *deadline*, may lead to catastrophic consequences; vehicle control systems are examples of such systems. On the other hand in *soft real-time systems* a number of deadlines can be missed without serious consequences. Examples of such systems are, e.g., multimedia systems.

A Real-Time System (RTS) consists of a number of *resources* (e.g. processors), a number of *tasks* (executing programs), designed to fulfil a number

of *timing constraints*, and a *scheduler* that assigns each task a fraction of the processor(s) time according to a *scheduling policy*. Tasks are *periodic* or *non-periodic*. Periodic tasks are an infinite sequence of *invocations* (task instances), while non-periodic tasks are invoked by external events. The choice of scheduling policy is made to satisfy the constraints imposed on the system. A task has timing constraints and timing properties. The timing constraints are imposed on the task and are, e.g., *deadline* (D) and *period / minimum interarrival time* (T / MINT). A task has timing properties, e.g., *worst-case execution time* (WCET) and in some cases *best-case execution time* (BCET). The WCET is the longest time it can take for a task to execute, and conversely the BCET is the shortest possible execution time. The Deadline is the longest allowed latency from the tasks nominal starting point of the task until its completion. This latency is dependent on when the task starts, and, e.g., how much it is interrupted by tasks with higher priorities. T is the nominal time between two task instances, and the MINT is the shortest time between two non-periodic task invocations. A transaction is defined as an ordered sequence of tasks to be executed in a specified order. A transactions has a period T_tr and is often constrained with an end-to-end deadline (E2ED).

Consider figure 1.1, in which task t_1 has a $WCET_1$, a $BCET_1$, a deadline DL_1 and a period T_1. A transaction tr_1 is defined over the tasks $< t_1, t_2, t_3 >$ and has an end-to-end deadline $E2ED_1$. The period of the transaction is decided by the first task in the transaction (t_1).

1.3 Component Based Development for Embedded Real-Time Systems

Component-Based Development, in general, is widely adopted in most mature engineering disciplines such as mechanics, electronics and construction. The component-based approach has also been used in computer science, within the development of desktop and internet applications, e.g., COM [10] and CORBA [11]. However, the component based strategy has not yet been as successful within embedded system software engineering as in the previously mentioned domains. The many reasons for this includes the varying demands of different domains in software engineering, unsatisfactory tool support [8] and lack of support for non-functional properties.

Further, desktop applications usually run on modern desktop computers with highly advanced processors, and huge amounts of memory, several hundreds of mega-bytes. Embedded systems on the other hand usually run on very

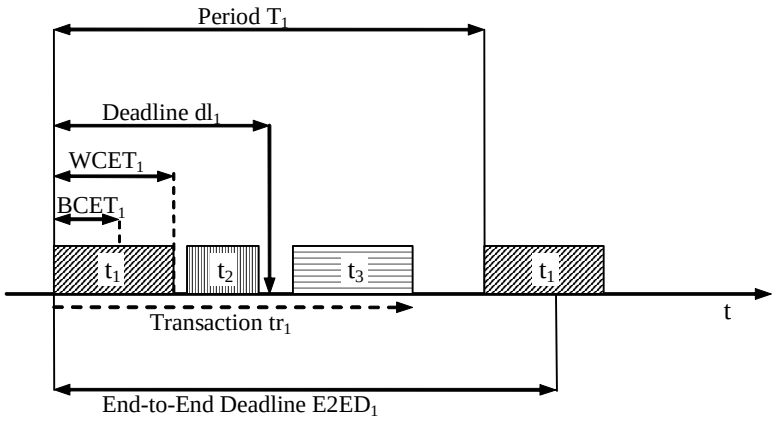

Figure 1.1: Real-Time Properties

small low-power processors, and often with no more memory than a few kilobytes. The difference in system characteristics calls for different approaches to fulfil the system requirements. For instance, desktop applications often do not require memory usage to be kept at a low level.

1.3.1 Component-Based Software Engineering Terminology

The basis of component-based systems is naturally the *component*. A software component is a software entity that conforms to a *component model* and can be *composed* without modification [2]. The term *component model* embraces the specification of components, how components are assembled, and the component framework. With other words, the component model is a set of rules governing how the components may or may not be used. The *composition* of components is the process of assembling components to form an application. Components are composed by constitute systems by connecting their interfaces according to the rules defined in the component model. The component interface is the entry to the component functionality. A component composition is executed in the context of a *component framework*. The component framework provides the necessary run-time support that is not provided by the underlying run-time system, e.g., scheduling, and finally, a *component technology* is the concrete implementation of a component model.

1.3.2 Component Based Development for Embedded Real-Time Systems

There have been many attempts to develop component models for embedded systems [12, 9, 13, 14, 15]. They have often been tightly coupled to a specific operating system or a specific domain and they seldom consider non-functional properties. Thus, they have not been general enough to be adopted for use in other domains [16]. Even though generality has not often been the goal of these models; in order for CBSE to be widespread in the embedded domain one or more standards must be developed. Standards for embedded CBSE will facilitate the integration of third party components and have shown to be effective in desktop applications.

Existing component technologies for embedded systems often do not provide support for non-functional properties. However, with a suitable resource-efficient component-technology with effective tools and support for non-functional properties, a more structured development can be incorporated in the embedded domain. Also, since product lines [17] are common within the domain, issues of commonality and reuse are central for reducing cost as well as increasing reliability.

It is, however, no more likely that one component-model will be applicable in all of software engineering, than that bridges and houses will be built from the same building blocks.

1.3.3 CBSE for vehicular systems

Vehicular systems incorporate highly advanced distributed embedded control systems. For instance, the luxurious new BMW [18] 7-series incorporates more than 65 *Electronic Control Units* (ECUs)

An ECU is an on-board computer system that runs software to control a specific process in the vehicle, e.g., the breaking system. Such systems are already component-based on a hardware level and the incorporation of component-based software systems would appear to be a natural abstraction.

In [19] the authors indicate that the costs of developing the electronics and software incorporated in a motor vehicle constitute more than half of the total development costs of the vehicle. Further requirements of the end-users mean that there is a continuous and increasing demand for both the hardware and software of vehicular electronics of increasing complexity. The objectives of CBSE in developing ECU's for vehicular systems are on the hand to lower the production costs, in terms of both development and hardware costs, and on the

other hand to cope with the increasing complexity of the emerging systems. CBSE has been shown to be an effective approach in lowering development costs and enhancing, e.g., reuse of components in other domains.

Component-based development of embedded vehicular systems usually requires a system architecture different from that of other systems. Due to the control nature of these systems, pipe-and-filter or blackboard architectures are usually used. In less restricted systems, such as multimedia systems, client-server architectures are used, such as the Koala component model for television sets [9]. Due to the difficulty of verifying the timing behaviour of client-server architecture it is not used in connection with vehicular embedded systems.

Moreover, middlewares and component frameworks must be deterministic in terms of timing behaviour and use a minimum of resources.

Analysis of the systems is essential to be able to prove that the vehicle is safe. Non-functional properties are very important as is tools and methods to be used by those performing analysis.

1.4 Outline of the thesis

The remainder of part I is organized as follows. Chapter 2 describes the Research Problems to be addressed and the questions derived. Chapter 3 describes the research performed and the answers obtained. Related work is described in chapter 4. Finally the thesis is concluded and future work is discussed in chapter 5. Part II contains all the papers included in chapters 6 to 10.

Chapter 2

Research Problems

In this chapter we will present the scope of our work by defining the research problems, introducing the research questions and presenting answers to these questions.

In the following section we will describe the problem we try to solve.

2.1 Problem Definitions

As mentioned in the introduction, component based development for embedded systems is not widespread. There are several presumptive reasons to this; there are no standard component models for embedded systems, thus it is hard to integrate third party functionality. The de-facto standard component models for desktop and internet applications are too resource demanding and they do not support non-functional properties. In order for a component model to be widely used within the embedded domain it has to have powerful design-time models and be efficient in terms of resource usage. With this in mind, we state two research problems, $P1$ and $P2$.

$P1$ *Today's component models do not consider adequately the transformation of design-time models to real-time models*. Most component models for embedded systems do not focus on low resource usage and analyzability and are developed for specific purposes. Any problems that arise are very specialized. Many qualities are implicit, or simply disregarded (e.g. real-time properties). Specifically we have observed that the mapping between components, and run-time entities, i.e., tasks, is totally omitted

from most existing component models, which leads to ineffective usage of the hardware. Moreover, it is in very few models that attention is given to optimization of performance and memory usage.

$P2$ *Today's component models do not have efficient resource usage*. When hard real-time properties are imposed on a system, the real-time properties are generally exaggerated to guarantee a predictable behaviour with respect to timeliness. The systems are highly complex and therefore there are very few ways of defining the non-functional properties in an analytically exact way. Properties such as, e.g., WCET, are increased to ensure that a deadline is not missed. As a consequence, the resource utilization is lower than the optimum.

For an overview of the problems stated, consider table 2.1. The table is used to identify more effectively similarities between the problems stated. The problems are summarized in the same order as they appear in the problem definition.

We can see that there are several issues in common to the problems stated. From an analysis of these issues we have derived two research questions, firstly one question with respect to resources, analyzability and mappings between components and tasks, the other one with respect to resource usage, efficiency and predictability.

Problem	**Problem Related Issues**			
$P1$	Resource usage	Analyzability	Mapping	Optimization
$P2$	Predictability	Resource usage	Efficiency	

Table 2.1: Summary of the core problems in the problem definition

2.2 Research Questions

As stated in the problem definition, resource usage and analyzability are important qualities in a component technology for embedded systems. One way of reducing resource usage is to map components to run-time tasks in such a way that memory usage and CPU-overhead is lowered. Components can be mapped to run-time tasks with respect to other properties as well. In order to do this, it is necessary to understand what qualities are important and how they are affected. Thus, technologies for embedded and real-time systems must define a set of non-functional properties. Non-functional properties are properties

that define different qualities of the software, e.g., timing behaviour, memory consumption, safety and reliability. These properties are for analysis purposes only and have no functional value. It is especially important to understand which non-functional properties are regarded as important by the industry, and which can be disregarded.

Thus, from these reflections we form our first question $Q1$:

What is required by a component technology for embedded systems in order to analyze mappings from design-time components to real-time tasks?

$(Q1)$

This question aims at understanding which requirements are important for a component technology for the vehicular domain regarding performing efficient mappings from design-time to run-time in component-based systems.

If a component technology supports the non-functional properties requested by the automotive domain, then how should they be realized? The properties define system qualities; hence, non-functional properties are only a measure of qualities of components or assemblies. What is really important is how the properties are realized. For instance, how is a component with *high reliability* implemented? In the case of component-based systems, how the components are assembled are equally important. Two components with *high reliability* individually cannot be guaranteed to provide *high reliability* when assembled [20] to form one system.

As stated earlier, most embedded systems in vehicles are real-time systems, and real-time systems are dependent on the timeliness; hence the real-time related non-functional properties must reflect the real-time behaviour. However, some real-time properties are hard to realize in an exact way. For instance the execution time of a complex system is difficult to establish because of the varying behaviour and the many program paths.

To guarantee timeliness in all possible circumstances predictions in real-time systems are based on worst-case scenarios. The actual-case is often lower than the worst-case due to safety margins, varying behaviour (e.g. different loop counts) leading to unused processor time, i.e., inefficient resource usage, and either simpler and less expensive hardware could have been used, or more functionality could have been implemented. However, the behaviour of software is often inherently variable, and to satisfy safety requirements, worst-case estimations should not be overly optimistic. How, therefore, can resource usage be reduced?

Component based systems often have very simple strategies for allocating components to run-time entities. In many component models there is no clear strategy for providing this mapping; thus this part is often not included in the component technology causing a gap between high-level design and concrete implementation. For CBSE to be adopted by domains using resource-constrained embedded real-time systems, the mapping between component models and real-time models must be automated and made mote efficient with respect to resource utilization. These reflections are summarized in question Q2:

How can resource efficiency and predictability be combined in a component model for embedded systems?

$(Q2)$

The answer to this question will be an explanation of how resource usage can be reduced through efficient component technologies, efficient mappings between component models and run-time systems; while maintaining the predictability of the system.

2.3 Contributions

The main contributions of the presented research have been summarized and are presented in the list below:

- A Classifications of the importance of software quality attributes according to companies in the vehicular domain.
- An Evaluation of the suitability of commercial component models with respect to the requirements of the vehicular industry.
- A proposal for the use of component to task allocation to increase performance and system utilization, and a proposal for an evaluation framework for such allocations.
- The development of a real-time component model that utilizes the *Multiple Versions Paradigm* together with different existing real-time scheduling methods and the *Adaptive Threshold Algorithm*.
- A proposal for the use of a software component technology for developing embedded systems through permitting the use of real-time theory by synthesis of run-time mechanisms for predictable execution according to the temporal specification of the component model.

The research performed and contributions made are further discussed in chapter 3.

Chapter 3

Research Contributions

Recent work [21, 22] has addressed the issue of reducing the cost of software development for embedded systems; specifically for automotive embedded systems. This thesis focuses on optimization of resource utilization when using component-based development. The research is a continuation of previous work [23].

We proceed from the research questions stated in chapter 2 by presenting the research topics in that order. The research is divided based on the questions Q1 and Q2. Each part answers parts of the questions. Finally we will present a discussion on the questions and the research topics. The research does not give complete answers to the questions, but will give partly answers and be subjects for further research.

The research has been performed in close cooperation with industry and academia. All research has been made in the context of two projects SAVE[1] and FLEXCON[2], but we have had close cooperation with other research groups, e.g., the HEAVE[3] project.

The following sections will describe each research topic.

[1] SAVE Project, http://www.mrtc.mdh.se/SAVE
[2] FLEXCON Project, http://www.mrtc.mdh.se/FLEXCON
[3] HEAVE Project, http://www.mrtc.mdh.se/HEAVE

3.1 Non-functional properties for embedded systems

Objective: The objective of this research is to find out which properties, both technical and development related, that are important for the vehicular industry and investigate how a number of, commercial and academic, component models fulfils these requirements. The purpose of the evaluations is to be a foundation for defining models, methods and tools for component-based software engineering. The objective is not to mutually rank component technologies or properties, but mainly point out what properties that are regarded as important by the industry within the vehicular domain, and investigate which properties that have good or poor support by existing technologies.

This research gives parts of the answer to question Q1.

Goal: This research aims at understanding which non-functional properties (quality attributes) are important for the vehicular domain, and show how well current component technologies fulfil these properties. We also present a list of quality attributes and analyze their impact in a component technology.

Research: The research is comprised by two parts; the first part is based on a survey that was sent to a number of representatives for different vehicular companies, and the second part is an evaluation of existing component technologies with respect to industrial requirements. In the first part, representatives from several companies were requested to prioritize a number of quality attributes *(extendibility, maintainability, usability, predictability, security, safety and reliability)* regarding importance. The representatives were requested to group the quality-attributes in four different categories *(very important, important, less important and unimportant)*. The list of quality attributes covers attributes that were thought to be of interest to the vehicular domain. The non-functional properties that we focus on most in this thesis are predictability regarding timeliness, and efficiency which are both considered as important properties. We also discuss the two most important properties safety and reliability.

The second part of this research is an evaluation of existing technologies. The technologies described and evaluated are PECT [12], Koala [9], Rubus CM [13, 24], PBO [14], PECOS [15] and CORBA based technologies [25]. The technologies were chosen firstly on the basis

that there is enough information available, and secondly that the authors claim that they are suitable for embedded systems. CORBA, however, was chosen as a reference technology to represent the desktop/Internet domain, although CORBA have flavours for small systems and for real-time systems.

The technologies were evaluated on how well they conform to twelve different, both technical and development related, requirements. The evaluation points out which requirements that are partly fulfilled by a component technology, and which are not. For instance the quality attribute *Testable and Maintainable* has received rather low points indicating that few component models have good support for these qualities. On the other hand most models have been considered *Understandable*. The requirements were gathered from an industrial case-study performed at Volvo Construction Equipment[4] and at CC-Systems[5]. The technologies evaluated in the second part of the research originate from both industry and academia. The evaluation work in this research has been workshop-oriented, where the authors have discussed and evaluated quality attributes and technologies.

Results: The research ended up with a grading of component models with respect to the studied requirements. The research shows that there is no single component technology that stands out as a very good candidate for fulfilling the industrial requirements from the second part of this research, nor the non-functional properties from the first part. Some properties are supported by very many technologies, while other properties are not considered at all. Component technologies that origin from academia puts more focus on extra-functional properties, while the industrial technologies are more pragmatic. Details about the research can be found in the included papers D and E.

Limitations and future work: There are several unanswered questions regarding the validity of the studies. Did the representatives of the companies represent the company or their own view of the study. Have all relevant representatives been interviewed, and are all relevant companies included in the process? The study should be extended to increase its validity and reliability. Although, unanswered questions, the study can

[4] http://www.volvo.com
[5] http://www.cc-system.se/

be used as a guide for coming studies, and as an indication to continued research.

3.2 Component models for embedded systems

Objective: Previous research has shown that there are component technologies that only partially support the properties that the vehicular domain finds important [26]. This research should aim at providing a component technology with powerful compile-time techniques and the desired support for the vehicular domain.

This research will give partly answers to question Q1 and Q2.

Goal: The research should propose the core of a component technology that utilizes powerful compile-time techniques and focuses on non-functional attributes and resource efficiency. The component technology should also be a foundation for defining new powerful compile time techniques.

Research: As described in the previous section, our research has pointed out that there are few component models that support some of the important non-functional attributes. We show how CBSE can be used for embedded real-time systems with high requirements on analyzability and low memory footprint. We consider the non-functional attributes predictability, reliability, safety and usability which are all expressed as important by the vehicular domain. Existing commercial component technologies often have powerful run-time mechanisms to realize the component-based approach, which is a disadvantage in terms of resource utilization for resource constrained systems. The idea of this research is to have expressive design-time models, utilizing the UML 2.0 standard, powerful compile-time techniques and efficient, temporally verified, mappings to a run-time system, e.g., a commercial real-time operating system. The defined component model is based on the pipe-and-filter interaction model and uses a *Read-Execute-Write* paradigm; all in-ports of a component are read, the component executes and finally writes all its out-ports. This execution model has the advantage of being highly analyzable. Moreover, the control systems in vehicles are often suitable for the pipe-and-filter paradigm. End-to-end deadlines are imposed to the model and are augmented with start and completion jitters. A middleware is proposed to handles all communication between the component

model and the underlying run-time system. This research is based on earlier work [27] and the experience from senior researchers.

Results: The research indicates that embedded component technologies can be resource efficient, reliable and easy to extend with support from powerful compile-time techniques. We have paid attention to expressed requirements and propose powerful compile time techniques and expressive design-time models. Details on the results can be found in included paper C.

Within the SAVE project this work has been extended with the implementation of the SaveCCM component technology [28].

Limitations and future work: The proposed techniques have not been industrially verified; although its successor (SaveCCM) has been implemented and verified in a small scale industry case-study within the SAVE project.

The model is restrictive and due to the pipe-and-filter interaction model it may not be suitable for some systems.

3.3 Allocating components to real-time tasks

Objective: Many component-based systems today use one-to-one allocations between design-time components and real-time tasks, or other rudimentary allocations. Finding allocations that co-allocate several components to one real-time task leads to better memory and CPU usage. However, the one-to-one allocations have the benefit of being highly analyzable, which is often a strong requirement in embedded systems, especially in embedded systems that handle time-critical functions such as engine-control and breaking systems.

A good allocation should be analyzable and reduce the amount of memory and CPU-usage compared to a one-to-one allocation.

This research answers parts of questions Q1 and Q2.

Goal: The research aims at finding near-optimal allocations that decrease memory and CPU usage, while preserving timeliness. Furthermore, the methods for allocating components to tasks should be general enough to be able to optimize regarding other properties besides memory and CPU-usage.

Research: The research is based on previous work [27, 29], which show that important issues in the embedded systems domain are predictability in terms of timeliness and performance in terms of memory consumption and CPU-load. To increase the performance and maintain the predictability we propose an allocation between components and real-time tasks, where several components are allocated to one task; thereby saving memory in terms of stack and task control blocks and CPU-time in terms of task switching.

Both the component model and task model used in this research is similar to those described in the previous section, where the run-time model uses tasks and transactions. The scheduling policy used is also fixed priority scheduling. A modified method of the Bate and Burns approach [30] is used to calculate the feasibility of the system in terms of real-time behaviour. The Bate and Burns approach assumes systems similar to the ones described in our research. They use fixed priority scheduling and transactions with end-to-end deadlines. We do not consider jitter and separation in our work, hence these requirements are disregarded. Moreover, they assume that a transaction has a period equal to the *least common multiple of all tasks participating in the transaction*. In our work we do not have this limitation. However, the approach is intuitive, and is easily adjusted to suit our needs.

Due to the combinatorial explosion of possible allocations from components to tasks, the problem is complex by nature. An allocation from components to a task is evaluated considering schedulability (timeliness) and *isolation*, where isolation is defined as *mutual exclusion of components regarding shared resources or other legitimate engineering reasons*. Because the problem is inherently complex the strategy is to evaluate our allocation approach by implementing a framework that utilizes a meta-heuristic search technique, in our case *Genetic Algorithms* (GA) [31]. GA can solve, roughly, any problem as long as there is some way of comparing two solutions. Each allocation is validated with respect to period-times, isolation, end-to-end deadlines and schedulability to ensure that an allocation is feasible. The proposed framework gives the possibility to optimize allocations regarding the properties memory consumption and CPU-overhead to find a *resource efficient* solution. Other possible approaches for solving the problem is different heuristics [32] or simulated annealing [33]. Genetic algorithms was chosen partly due to in-house experience, but also because it is very versatile and can han-

dle several dimensions as opposed to, e.g., simulated annealing.

Similar approaches, for the parallel and distributed domain has been performed, e.g., *Partitioning and Clustering*. Partitioning is a method for dividing an application into small blocks for execution on separate nodes. Clustering is the issue of allocating nodes of a task graph to labelled clusters, where the task graph represents the application behaviour. Compared to our research partitioning is the opposite, dividing applications for parallelism as opposed to co-allocating several components to one task considering speed and memory. Clustering allocated tasks to clusters regarding, mostly, parallelism. Clustering is based on task behaviour in form of directed acyclic task graphs, as opposed to our component-to-task approach. Clustering is often used for placing tasks on distributed nodes in a schedulable way. Our approach also regards timeliness, but performs the allocation regarding optimization of given non-functional properties, currently memory usage and CPU-overhead.

Results: The results from the evaluation were satisfactory, and we have found that for industrially representative systems memory consumption and CPU-overhead can be decreased by as much as 32% and 48% respectively compared to a one-to-one mapping. Details about the methods, framework and results can be found in the included paper A.

Limitations and future work: The research only consider *pipe-and-filter* and *blackboard* architectures. Further we do not consider blocking or advanced real-time properties such as jitter or separation. In practice, the allocation will probably have to be guided with a knowledge base or interactivity with engineers. Further, for validity reasons, the allocation should be verified with an industrial case-study.

Future work includes optimizing allocations regarding several non-functional properties.

3.4 Resource reclaiming

Objective: In real-time systems there are often unused resources in terms of CPU-time due to pessimistic predictions. These resources can be used for executing tasks, e.g., more often, or with higher quality (longer time).

This research will give partly answers to question Q2.

Goal: This research aims at defining methods for using residual time from pessimistic real-time predictions to provide a higher quality of service.

Research: We show how component technologies can be extended with multiple services to provide different quality levels depending on residual time in real-time systems. We do this by combining the multiple versions paradigm [34], with the adaptive threshold algorithm [35]. The multiple versions paradigm allows us to have several versions (services) of the same component. In this research the multiple versions is used for the same functionality with different quality; consider, e.g., more or less iterations in a numerical approximation. Thereby we provide different quality levels of the same component. Each quality level is associated with a value which is accumulated to the system as that quality level is chosen and executed. The Adaptive threshold algorithm allows us to provide a system that strives to maximize the total system value by choosing the appropriate quality level dependent on the residual time of the system.

However, the multiple versions and adaptive threshold algorithm generates some extra overhead in the system, both in terms of memory and CPU-time. Thus this approach may not be appropriate for very small systems with extreme requirements on keeping the memory and CPU-overhead low. The research is based on knowledge from senior researchers, and literature studies of previously published research. Although, by combining this approach with the previously described component to task allocation approach may be interesting future work.

Results: The research shows how the multiple versions paradigm and the adaptive threshold algorithm can be combined with the notion of residual time for providing higher quality of service in a system. The results from this research can be found in included paper B.

Limitations and future work: The proposed approach adds complexity to, in some sense, reduce resource usage. Further, the approach requires that a solution or method can be divided into several quality levels. Hence, it is highly suitable for, e.g., numeric approximations. However, all system may not easily be divided into quality levels. Finally, the value of each quality level is implicit, and must be acquired in some way.

3.5 Conclusion

In this section we connect the research questions with the described research topics.

Question $Q1$**:** *What is required by a component technology for embedded systems in order to analyze mappings from design-time components to real-time tasks?* From the research summary, we can see that this question is answered by the first three research topics. The first, which investigates how well current component technologies support non-functional properties, and the second and third that investigates which requirements are important for lowering resource usage and performing good allocations between components and tasks.

Question $Q2$**:** *How can resource efficiency and predictability be combined in a component model for embedded systems?* We note that three last research topics contribute with answers to this question. The second research topic considers resource efficiency and predictability by proposing efficient compile-time techniques for embedded component technologies.

The third considers resource efficiency by proposing mappings between components and tasks through using stochastic search algorithms to find near optimal solutions. The fourth and last research topic considers resource management and predictability in terms of timeliness by incorporating the multiple versions paradigm with the adaptive threshold algorithm together with fixed priority real-time analysis.

Hence both questions Q1 and Q2 have been at least partly answered. We have of course only provided one possible answer to each question, where many more answers are feasible.

Chapter 4

Related Work and Basic Principles

In this section we will explore research areas related to the research described in chapter 3. We will focus on the areas *Real-Time Analysis*, *Task Allocation Optimization Techniques* and *Component Models*.

Real-Time Analysis is an area where much research has been conducted. In this section we will discuss the classical exact analysis, transactions, attribute assignment and resource management.

4.1 Real-time systems

In the past decades a lot of research has been performed within the domain of real-time systems. A majority of the research is aimed at schedulability analysis and system predictability. As early as 1973, Liu and Layland published work on real-time analysis and scheduling [36], defining the *earliest deadline first* (EDF) and *rate monotonic* (RM) scheduling policies. Since then, a myriad of different scheduling techniques has been produced. The scheduling policies can be divided into three paradigms, these are:

- Priority-driven (e.g., RM or EDF) [36]
- Time-driven (table-driven) [37, 38]
- Share-driven [39]

The industry today primarily uses priority-driven scheduling (e.g. Rubus blue part and VxWorks) [40, 41]. A few companies use time-driven off-line scheduling and cyclic scheduling.

Real-Time Systems consists of a number of resources, e.g., scheduler and CPU. The scheduler decides which task is assigned CPU-time, and for how long. A task is a schedulable entity that consists of a control block with data for the scheduler and operating system such as period, priority and pointers to user code and user data.

4.1.1 Classical Real-Time Analysis

A schedule has a fix time LCM (Least Common Multiple) time units, thus the schedule is repeated every LCM time units. Hence, if one instance of the schedule over the time $[0 - LCM)$ is feasible, then all instances of that schedule will be feasible.

For time-driven scheduling a schedule is created off-line (pre run-time). The schedule is created according to some heuristics to ensure that it is feasible.

For priority-driven scheduling, the feasibility can be analytically calculated by analyzing the longest time a task can be pre-empted (interrupted) by other tasks to decide its longest response time. A task in a priority driven system has at least three analytical properties:

- Period time (how often the task is invocated)
- Worst-case execution time *WCET* (how long the task runs)
- Deadline (when the task must be finished)

and

- Priority (how important is the task)

There are both fix priority scheduling and dynamic priority scheduling. As the names reveal, a task in a fix priority schedule is assigned a priority that never changes. Examples of fix priority scheduling policies are *Rate Monotonic*, where the priority is the inverted period time, and *Deadline Monotonic*, where the priority is the inverted deadline.

When using Dynamic priority scheduling, the priorities change during run-time. An example of a dynamic priority scheduling policy is *earliest deadline first* (EDF).

There are both pros and cons with both these approaches. Changing priorities dynamically during run-time requires computations, thus generating overhead. However, it can be assured that the task that is in greatest need of executing gets CPU-time. For instance, the EDF scheduling policy is optimal, i.e., there is no scheduling policy that is better; although, then the overhead is not considered.

In this section we will concentrate on fix-priority scheduling.

A scheduler decides, based on the priorities of the tasks, what task should run next. There are two different approaches on how to handle this, besides the scheduling policy; pre-emptive scheduling, and non-pre-emptive scheduling. In pre-emptive systems, a task with a higher priority will interrupt a lower priority task, if the higher priority task is scheduled to run. In a non-pre-emptive system, a task can not be interrupted but always finishes its execution. Pre-emptive systems have the advantage of being more dynamic, and provides shorter response-times for time-critical tasks. However, the complexity of the system and the schedule is also increased.

The most commonly used analytical approach for priority-driven systems is the *exact analysis*. Exact analysis analyses the feasibility of the task-set considering scheduling. We will show the classical exact analysis for pre-emptive systems.

Exact Analysis

Liu and Layland [36] present analysis to calculate the worst-case response time R_i. The response time R_i for task t_i has to be less or equal to the deadline D_i. The deadline is assumed to be equal to the period time. The analysis begins with the highest priority task. Then the exact analysis is defined as:

$$R_i = WCET_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil WCET_j \tag{4.1}$$

Where hp(i) is the set of tasks with higher priority than i.

Equation 4.1 is solved by forming a recurrence relationship:

$$W_i^{n+1} = WCET_i + \sum_{j \in hp(i)} \left\lceil \frac{W_i^n}{T_j} \right\rceil WCET_j \tag{4.2}$$

The set of values W_i^0, W_i^1,...,W_i^n is monotonically none decreasing. When $W_i^n = W_i^{n+1}$ the solution to the equation has been found. If all tasks passes

the test they will meet all their deadlines; if they fail the test then, at run time, a task will miss its deadline (unless the worst-case execution time estimation turn out to be too pessimistic).

4.1.2 Transactions

Transactions are collections of related tasks, which collectively perform some function or have some shared timing attributes. A transaction tr usually has a timing requirement, i.e., an end-to-end deadlines D_tr. The transaction usually has a period T_tr which denotes a lower bound on the time between re-arrivals of the transaction. Unfortunately exact analysis is computationally infeasible to evaluate for task-sets with transactions[42]; hence some other approach has to be used. To use exact analysis the schedule has to be simulated over the *hyper-period* (LCM of all periods). A common approach is schedule simulation [43]. In [44] Redell presents a fast method for calculating response-times for task sets with transactions with advanced timing-properties such as offsets and jitter.

4.1.3 Attribute Assignment

For a real-time schedule to be feasible, the task attributes have to be set accordingly. Several publications exist on the matter but many of them, e.g., [45, 46], are not very straight forward and difficult to use. That is because they are difficult to justify and they assume that all attributes are changeable. A more straight forward approach is the one by Bate and Burns [30]. Timing requirements such as *Period, Deadline, Jitter* and *Separation* are considered. Furthermore, transactions are sequences of tasks executing in a fixed order. The timing requirements for transactions are *Period, End-to-End Deadlines* and *Jitter*. Bate and Burns use an iterative approach by considering subsequent instances of tasks within one transaction and derive the attributes from the iterative process. Their approach is somewhat similar to schedule simulation.

4.2 Resource Management

Hard Real-Time Analysis is pessimistic because of the fact that the timing criteria must always be fulfilled in any situation. Assume a program that has more than one path and both paths take different long to execute. Then the program must be assigned a worst-case execution time that corresponds to the longer path. If, in fact, the longer path is run very seldom, there will be a

lot of capacity in vain. Several approaches have been suggested to use this extra time (residual time). The slack-stealing algorithm [47] is an algorithm that tries to use as much residual time as possible for, e.g., serving aperiodic requests. Other approaches include the constant bandwidth server [48] and the total bandwidth server [49]. An approach to decide what service should be executed in the residual time is the adaptive threshold algorithm [35]. The Adaptive Threshold Algorithm decides based on values assigned to every pending service, aperiodic or periodic, which service should be executed. The goal of the algorithm is to maximize an accumulated value.

4.3 Task allocation

Allocation from components to real-time tasks is a subject that has not been widely explored. Several related allocations in parallel and distributed systems exist, e.g., partitioning and clustering, and we will discuss these.

Tasks allocation is the issue of allocating a task to different resources. A common research area is to allocate tasks to processors in a multi processor system. Different approaches have been used [50, 51, 52, 53, 54].

4.3.1 Task to Node Allocation in Distributed Systems

Tindell et.al [51] uses a stochastic search method to allocate tasks in a distributed real-time system. The majority of these approaches focus on load-balancing in order to minimize communication overhead. In [55] genetic algorithms and simulated annealing is used to select topology and place tasks on nodes in distributed control systems.

4.3.2 Partitioning

An other approach to task allocation for parallel or multi-threaded systems is the partitioning [56] approach. Partitioning is the issue of that, for an application to operate on a multiprocessor system, it must be divided into separate threads of execution for each processor. Sarkar [57] describes a system for automatically partitioning and scheduling parallel applications on multiprocessor systems. SarkarŠs approach is dividing the applications to the smallest possible fundamental blocks, and then merges them back until the number of blocks left equals the number of processors.

In [58, 59] partitioning is used to divide a control application into processes for semi-distributed real-time control systems.

4.3.3 Clustering

To efficiently execute programs in parallel on a multiprocessor system, a minimum execution time multi-processor scheduling problem must be solved to determine the assignment of tasks to the processors and the execution order of the tasks so that the execution time is minimized [60]. Clustering [61] is the issue of mapping of the nodes of a task graph onto labelled clusters. A cluster consists of a set of tasks and each task is an indivisible unit of execution. Each cluster executes on a separate processor. In [62] a partitioning and scheduling technique for streamlining inter-process communication is suggested. Multiprocessor clustering techniques are exploited to increase the compile-time tolerance of the embedded systems domain. The authors in [62] suggest methods for efficient mapping of applications to multiprocessor architectures by using clustering.

4.4 Optimization of real-time systems

Different optimization techniques have regularly been used to find solutions for complex problems. The problem of finding allocations between component models and real-time models is a problem that grows very rapidly in terms of possible solutions. To find a solution within a reasonable time optimization techniques are used.

Different optimization techniques have been used to solve different problems, in, e.g., [63] genetic algorithms are used for assigning attributes for complex real-time constraints. In [51] simulated annealing is used for assigning tasks to different nodes in a distributed real-time system.

4.4.1 Heuristic methods

Due to the difficulty to find optimal solutions to allocation problems, heuristic methods are commonly adopted. A heuristic method in this context is a method that uses some rules to create a solution. These rules are defined by the algorithm designer who uses intuition and experience. Example of commonly used heuristics for, e.g., the classical bin-packing problem is *first fit*, *best fit* and

worst fit. Which heuristics that fits a specific problem is hard to tell in advance, hence experience or in-depth knowledge of the problem is often required.

4.4.2 Genetic Algorithms

A GA is a guided search technique, based on models of Darwinian [64] and Lamarkian [65] evolution. Solutions are represented by fixed length strings. The value of a solution is a measure of its ¨ fitness for purpose¨ . Many researchers have applied machine-learning methods to solve or optimize different problems. Genetic algorithms [66, 67] were formally introduced in the United States in the 1970s by John Holland [31] at the University of Michigan. Genetic Algorithms, together with, e.g., simulated annealing [33, 68], belongs to the class of stochastic search methods.

The continuing price/performance improvements of computational systems have made stochastic search techniques attractive for different types of optimization. In particular, genetic algorithms work very well on mixed (continuous and discrete), combinatorial problems. They are less susceptible to getting 'stuck' at local optima than gradient search methods. But they tend to be computationally expensive.

In order to use genetic algorithms, the problem must be represented as a genome (chromosome). The GA then creates a population of solutions and applies genetic operators such as mutation and crossovers to find the best solution. A mutation is random bit flips, and a crossover is defined so that two individuals (genomes) combine to produce two new individuals (children). There also exist asexual crossovers, or single-child crossovers.

GA uses a direct analogy of natural behaviour. The salient features of each individual population member are represented by a string, referred to as a *chromosome*[1]. The components of the strings are called *gene*. When using GA the design problem often have to be represented by bit strings.

4.4.3 Simulated Annealing

Simulated Annealing (SA) [33] is a variant of the stochastic search methods, and has been applied to a wide range of practical problems. SA was initially inspired by the laws of thermodynamics which state that at temperature, t, the

[1]Chromosome stands for coloured body after the colouring of nuclei in early experiments to identify DNA

probability of an increase in energy of magnitude, δE, is given by equation 4.3

$$P[\delta E] = exp(-\delta E/kt) \tag{4.3}$$

Where k is the physical constant known as *Boltzmann's constant* and t can be considered to b a parameter of the process. In a simulated version this equation is used within a system that is 'cooling' towards a steady state. SA can find solutions in non-linear models. It is versatile since it does not rely on any restrictive properties of the model.

In order to use SA a representation of possible solutions and an *annealing schedule* (an initial temperature and rules for lowering it) are required.

4.4.4 Branch-and-Bound

Branch and Bound (B& B) [69] is a widely used optimization and search method for solving complex discrete optimization problems. It belongs to the class of *implicit enumeration* methods, meaning that it make a limited enumeration of possible solutions in order to find an optimal or sub-optimal solution. The algorithm partitions the total solution domain into smaller and smaller subsets, thereby the name *branch*. After the branching, each subset is assigned a value, and thereby some subsets can be eliminated from further consideration (there by *Bounding*).

4.5 Component models for embedded systems

We will discuss component models regarding their mapping to run-time systems and their relation to non-functional properties. We divide the component models in categories relating to the application domain of our research, i.e., *Vehicular systems, Consumer Electronics* and *Industrial systems*.

This section of related work is based on the State Of The Art report (SOTA) [70] that was produced as a pre-study to our research.

In this section we will discuss component models used, or suitable, for different domains. We will discuss the application areas previously defined, i.e., vehicular systems, consumer electronics and Industrial systems.

4.5.1 Component models for Vehicular Systems

Component models for vehicular systems have high requirements on reliability and predictability. One component model that is successfully used within the automotive domain is the Rubus Component Model.

Rubus Component Model

The Rubus Component Model (Rubus CM) [13] is developed by Arcticus systems. The component technology is tailored for resource constrained systems with real-time requirements. The Rubus Operating System (Rubus OS) [24] has one time-triggered part (used for time-critical hard real-time activities) and one event-triggered part (used for less time-critical soft real-time activities). However, the Rubus CM is only supported by the time-triggered part.

The Rubus CM runs on top of the Rubus OS, and the Rubus CM is tightly coupled to the Rubus OS. The Rubus OS is very small, and all component and port configuration is resolved off-line by the Rubus configuration compiler.

Non-functional properties can be analysed during design time since the component technology is statically configured, but timing analysis on component and node level (i.e. schedulability analysis) is the only analysable property implemented in the Rubus tools.

The Rubus component model has a rather rudimentary mapping from components to tasks. All components are scheduled off-line and are then basically assigned one task each.

Thus, Rubus does not fully consider the requirements stated by the vehicular domain due to the rudimentary mapping to real-time tasks, and the few non-functional attributes.

SaveCCM Component Model

The SaveCCM Component Model [28] is developed at the Mälardalen Real-Time Research Centre within the bounds of the SAVE[2]-Project. The component model is tailored for resource constrained systems with real-time requirements. Unlike Rubus CM, SaveCCM support a variety of different non-functional properties. The properties are analyzed during design-time, and the technology is statically configured at compile-time. SaveCCM is not bound to any operating system but generates intermediate code that can be translated to specific programming languages.

SaveCCM has is built on the pipe-and-filter interaction model, and separates data, control and analytical interfaces. There is a construction called switches that acts as configuration mode changes pre-run-time, and acts as logical conditions between interfaces during run-time. The component technology is augmented with transactions and end-to-end deadlines. A switch can also split and join transactions.

[2] http://www.mrtc.mdh.se/SAVE

SaveCCM uses simple heuristics to map components to tasks by assigning all components that belong to the same transaction and are not separated by any switch.

SaveCCM is probably the best suited technology considering the requirements from the vehicular domain. It considers our requirements regarding non-functional attributes. However, the mapping between components and real-time tasks can be improved.

4.5.2 Consumer Electronics

Koala

The Koala component technology [9] is designed and used by Philips for development of software in consumer electronics. Typically, consumer electronics are resource constrained since they use cheap hardware to keep development costs low. Koala is tailored for Product Line Architectures [71]. The Koala components can interact with the environment, or other components, through explicit interfaces. The interfaces are statically connected at design time.

Low resource consumption was one of the requirements considered when Koala was created. Passive components are allocated to active threads during compile-time and they interact through a pipes-and-filters model. A construction called thread pumps is used to decrease the number of processes in the system. Koala does not support analysis of run-time properties; however, research has presented how properties like memory usage and timing can be predicted in general component-based systems, although the thread pumps used in Koala might cause some problems to apply existing timing analysis theories. Furthermore, Koala is implemented for a specific operating system.

Since Koala uses thread pumps it is difficult to analyze the mapping between the components and the run-time system. Because Koala lacks non-functional properties it is not very predictable.

Koala does not consider any of the requirements from the vehicular domain; not non-functional properties, nor predictable mappings to real-time tasks.

Robocop

Robocop is a component model developed in Eindhoven University. It is a follow-up, or a variant of, the Koala model. The aim of Robocop is to define an open component-based framework for the middleware layer in high volume embedded applications [72]. A component framework and component models in different abstractions form the core of the Robocop architecture. Un-

like Koala, Robocop has several non-functional properties such as timeliness, performance, reliability, availability, safety and security. The model is based on resource predictions, which does not give 100% guarantees unlike formal methods. Therefore, it can not be considered suitable for safety-critical systems.

A Robocop component is a set of models providing information about the component. The models may be in different forms; human readable form, e.g., documentation or binary form. Other types of models are functional model and non-functional model where the functional model describes the functionality of the component, whereas the non-functional model describes timing, reliability, memory usage etc.

In [73, 74] an approach to define tasks from component behaviour is suggested. The tasks are assigned non-functional properties, and are analyzed regarding schedulability in order to guarantee a feasible task set. However, there is no attempt to optimize the allocation regarding any property.

4.5.3 Industrial Systems

PECOS

PECOS (PErvasive COmponent Systems) [15, 75] is a collaborative project between ABB Corporate Research Centre and academia. The goal for the PECOS project was to enable component-based technology with appropriate tools to specify, compose, validate and compile software for embedded systems. The component technology is designed especially for field devices, i.e. reactive embedded systems that gathers and analyse data via sensors and react by controlling actuators, valves, motors etc.

Non-functional properties like memory consumption and worst-case execution-times are associated with the components. These are used by different PECOS tools, such as the composition rule checker and the schedule generating and verification tool.

The PECOS component technology uses layered software architecture. One of the layers is the Run-Time Environment (RTE) that takes care of the communication between the application specific parts and the real-time operating system. The components communicate using a data-flow-oriented interaction, it is a pipes-and-filters.

The PECOS architecture does not handle the actual mapping from components to tasks. A Run-Time Environment Layer is defined to communicate with the underlying real-time operating system. However, the mapping is left

to the policy of the operating system.

The PECOS component technology has a lot of focus on non-functional properties, which goes in line with the requirements stated by the companies in the vehicular domain. The RTE gives the possibility to map PECOS components to several platforms. However, the component technology does not consider this mapping. Though, the task allocation approach suggested in this thesis can be used for resource efficient mappings.

PBO

Port Based Objects (PBO) [14] combines object oriented design, with port automaton theory. PBO was developed as a part of the Chimera Operating System (Chimera OS) project [76], at the Advanced Manipulators Laboratory at Carnegie Mellon University. Together with Chimera, PBO forms a framework aimed for development of sensor-based control systems, with specialisation in reconfigurable robotics applications. An explicit design goal for a system based on PBO was to minimise communication and synchronisation, thus facilitating reuse.

PBO implements analysis for timeliness and facilitates behavioural models to ensure predictable communication and behaviour. The communication and computation model is based on the pipes-and-filters model.

The Chimera OS is a large and dynamically configurable operating system supporting dynamic binding, it is not resource constrained. The low coupling between the components makes it easy to modify or replace a single object. Due to the low coupling between components through simple communication and synchronisation the objects are highly reusable. The maintainability is also affected in a good way due to the loose coupling between the components.

A single PBO-component is tightly coupled to the Chimera OS, and is an independent concurrent process, i.e., they use a rudimentary one-to-one mapping.

This component technology has too little focus on non-functional properties considering the requirements on vehicles. Further the mapping between components and tasks is a one-to-one mapping, where resource efficiency is not considered.

IEC 61131 : Programmable Logic Controllers

Because of the lack of standards for PLCs (Programmable Logic Controllers), IEC instituted this standard in 1993 [77]. At that time several well established

techniques for programming PLCs existed, so the authors of the standard found it necessary to include several different programming methods. The standard describes three graphical and two text based languages, it concentrates on the syntax and leave the semantics less definitive. It is port-based and very resource constrained. There are no non-functional properties which makes the analyzability low.

One of the graphical languages included in the standard can be called a component language, the Function Block Diagram (FBD) language. It is a graphical language that can be used to define applications in terms of control blocks, which also can be imagined as components.

In principle, user-defined components (function blocks) may contain control code very similar to conventional PLC programs in any of the defined programming languages. The control code defined in the components can therefore be re-used within the same PLC task, shared between multiple tasks. There is however no pronounced strategy for allocating components to tasks, and it is up to the user to define an allocation.

Further, IEC61131-3 does not support non-functional properties. Thus, it is not very suitable regarding vehicular requirements.

Chapter 5

Conclusions and Future Work

We have addressed the problem of mapping components to run-time systems by developing answers to two main research questions. One of the main contributions of the thesis is the development and evaluation of the methods used in allocating components to real-time tasks. The evaluation clearly shows that these methods can provide substantial benefits in terms of reduced memory consumption and CPU-utilization and thereby in terms of a reduced hardware requirement. This is the main contributor to answering the questions. We have also investigated several issues concerning component model requirements in relation to the allocation of components. We have also investigated several issues concerning what quality attributes are important for the vehicular industry. Further we have studied resource effective component technologies and resource reclaiming for the often pessimistic real-time analysis. In the following sections we will summarize the contributions of this work, and discuss how our research will be continued in the future.

5.1 Summary

The main contributions of the presented research have been summarized and are presented in the list below:

- A Classifications of the importance of software quality attributes according to companies in the vehicular domain.

- An Evaluation of the suitability of commercial component models with respect to the requirements of the vehicular industry.
- A proposal for the use of component to task allocation to increase performance and system utilization, and a proposal for an evaluation framework for such allocations.
- The development of a real-time component model that utilizes the *Multiple Versions Paradigm* together with different existing real-time scheduling methods and the *Adaptive Threshold Algorithm*.
- A proposal for the use of a software component technology for developing embedded systems through permitting the use of real-time theory by synthesis of run-time mechanisms for predictable execution according to the temporal specification of the component model.

5.2 Future Work

Future work will primarily be in a further study of the allocation from components to real-time tasks. It will include the addition of other allocation criteria, e.g., by adding jitter and blocking requirements. When adding jitter constraints and blocking, trade offs arise between switch overhead and memory size versus deviation from nominal start and end times and blocking times. Furthermore, a more efficient scheduling policy and priority assignment will be applied. Due to the architecture of the GA it is easy to add new optimizations such as those proposed.

Further validation of the work presented in this thesis is necessary. In order to facilitate this, a prototype implementation of a component technology within the SAVE project is under development where the core part is being completed. The prototype will enable evaluation of different technology realisations with respect to performance. The model transformation of that technology needs additional attention, particularly the strategies for allocation of components to tasks. We will integrate our methods into this component technology (SaveCCM [28]).

Other approaches to future work are to add Case-Based Reasoning (CBR) for a knowledge based approach to help engineers determine suitable task allocations for specific domains.

Bibliography

[1] C. Szyperski. *Component Software - Beyond Object-Oriented Programming*. ISBN 0-201-74572-0. Addison-Wesley, 1998.

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

[3] M. D. McIlroy. Mass produced software components. In *Proceedings of Nato Software Engineering Conference*. NATO, 1968.

[4] J. Turley. The two percent solution. Technical report, Embedded Systems Programming, http://www.embedded.com/showArticle.jhtml?articleID=9900861. (Last Accessed 2005-02-17), December 2002.

[5] IEEE. The new ieee standard dictionary of electrival and electronics terms. Technical report, 1992.

[6] J. A. Stankovic and K. Rammaritham. Hard real-time systems. *IEEE Computer Society Press, Washington, D.C., USA*, 1988.

[7] Autosar project. http://www.autosar.org/ (Last Accessed: 2005-03-15).

[8] I. Crnkovic. Component-based approach for embedded systems. In *Ninth International Workshop on Component-Oriented Programming, Oslo*, June 2004.

[9] R. van Ommering, F. van der Linden, and J. Kramer. The koala component model for consumer electronics software. In *IEEE Computer*, pages 78–85. IEEE, March 2000.

[10] D. Box. *Essential COM*. ISBN 0-201-63443-5. Addison-Wesely, 1998.

[11] OMG. Corba component model 3.0. Technical report, OMG Documatation (ccm/02-04-01), April 2002.

[12] K. C. Wallnau and J. Ivers. Snapshot of ccl: A language for predictable assembly. Technical report, Software Engineering Institute, Carnegie Mellon University, 2003. CMU/SEI-2003-TN-025.

[13] K. L. Lundbäck. Rubus os reference manual. general concepts. arcticus systems: http://www.arcticus.se.

[14] D. B. Stewart, R. A. Volpe, and P. K. Khosla. Design of dynamically reconfigurable real-time software using port-based objects. In *IEEE Transactions on Software Engineering*, pages 759–776. IEEE, December 1997.

[15] M. Winter, R. Genssler, A. Christoph, O. Nierstraszn, S. Ducasse, R. Wuyts, G. Arevalo, P. Muller, C. Stich, and B. Schönhage. Components for embedded software . the pecos apporach. In *Proceedings of Second International Workshop on Composition Languages In conjunction with 16th European Conference on Object-Oriented Programming (ECOOP) Malaga, Spain*, June 2002.

[16] A. Möller, M. Åkerholm, J. Fredriksson, and M. Nolin. Software component technologies for real-time systems - an industrial perspective. In *WiP Session of Real-Time Systems Symposium (RTSS) Cancun, Mexico*, December 2003.

[17] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2001. ISBN 0-201-70332-7.

[18] BMW. Bmw homepage : http://www.bmw.com.

[19] N. Andersson. Halva bilens värde är elektronik. Technical report, September 2002.

[20] M. Larsson. *Predicting Quality Attributes in Component-based Software Systems*. PhD thesis, Mälardalen University, March 2004.

[21] M. Åkerholm. A software component technology for vehicle control systems - trade-off between enginnering flexibility and predictability. Technical report, Technology Licentiate Thesis No.26, ISSN 1651-9256, ISBN 91-88834-XX-X, Mälardalen Real-Time Reseach Centre, Mälardalen University, February 2005.

[22] A. Möller. Software component technologies for heavy vehicles. Technical report, Technology Licentiate Thesis No.26, ISSN 1651-9256, ISBN 91-88834-88-3, Mälardalen Real-Time Reseach Centre, Mälardalen University, January 2005.

[23] C. Norström, M. Gustafsson, K. Sandström, J. Mäki-Turja, and N. Bånkestad. Experiences from introducing state-of-the-art real-time techniques in the automotive industry. In *In Eigth IEEE International Conference and Workshop on the Engineering of Compute-Based Systems Washington, US*. IEEE, April 2001.

[24] K. L. Lundbäck, J. Lundbäck, and M. Lindberg. Component-based development of dependable real-time applications, 2003.

[25] OMG. The common object request broker: Architecture and specification. Technical report, OMG Formal Documatation (formal/01-02-10), February 2001.

[26] A. Möller, M. Åkerholm, J. Fredriksson, and M. Nolin. Evaluation of component technologies with respect to industrial requirem. In *Euromicro Conference, Component-Based Software Engineering Track Rennes, France*, August 2004.

[27] O. Bridal C. Norström S. Larsson H. Lönn M. Strömberg H. Hansson, H. Lawson. Basement: An architecture and methodology for distributed automotive real-time systems. *IEEE Transactions on Computers*, 46(9):1016–1027, Sep 1997.

[28] H. Hansson, M. Åkerholm, I. Crnkovic, and M. Törngren. Saveccm - a component model for safety-critical real-time systems. In *Euromicro Conference, Special Session Component Models for Dependable Systems Rennes, France*. EEE, September 2004.

[29] K. Sandstrom, J. Fredriksson, and M. Åkerholm. Introducing a component technology for safety critical embedded real-time systems. In *Proceeding of CBSE7 International Symposium on Component-based Software Engi-neering*. IEEE, 2004.

[30] A. Bate and I. Burns. An approach to task attribute assignment for uniprocessor systems. In *Proceedings of the 11th Euromicro Workshop on Real Time Systems, York, England*. IEEE, June 1999.

[31] John H. Holland. Genetic algorithms and the optimal allocation of trials. *SIAM J. Comput.*, 2(2):88–105, 1973.

[32] Kenyon. Best-fit bin-packing with random order. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 1996.

[33] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671–680, 1983.

[34] J. A. Stankovic and K. Ramamritham. What is predictability for real-time systems? *Real-Time Systems*, 2(4):247–254, November 1990.

[35] N. Audsley R. Davis, S. Punnekkat and A. Burns. Flexible scheduling for adaptable real-time systems. *Proceedings of IEEE Real-Time Technology and Applications Symposium*, May 1995.

[36] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in hard-real-time environment. *Journal of the Association for Computing Machinery (ACM)*, 20(1):46–61, 1973.

[37] H. Kopetz. The time-triggered model computation. In *Proceedings of the 19th IEEE Real-Time Systems Symposium (RTSS'98),Madrid, Spain*, pages 168–177, December 1998.

[38] C. W. Hsueh and K. J. Lin. An optimal pinwheel sceduler using the single number reduction technique. In *Proceedings of the 17th IEEE Real-Time Systemss Symposium (RTSS'96), Los Alamitos, CA, USA*, pages 196–205, December 1996.

[39] I. Stoica, H. Abdel-Wahab, K. Jeffay, S. Baruha, J. Gehrke, and G. Plaxton. A proportional share resource allocation algorithm for real-time, time-shared systems. In *Proceeding of 17th IEEE Real-Time Systems Symposium (RTSS'96), Los Alamitos, CA, USA*, pages 288,199. IEEE, December 1996.

[40] Wind River. Wind river homepage: http://www.windriver.com.

[41] Arcticus. Arcticus homepage: http://www.arcticus.se.

[42] K. Tindell, H. Hansson, and A. Wellings. Analyzing real-time communications: Controller area network (can)). In *Proceedings of Real-Time Systems Symposium, RTSS*, pages 259–263, 1994.

[43] N. C. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical report, Technical Report, Department of Computer Science, University of York, 1991.

[44] O. Redell and M. Törngren. Calculating exact worst case response times for static priority scheduled tasks with offsets and jitter. In *Proc. Eighth IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE, September 2002.

[45] R. Gerber, S. Hong, and M. Sabsena. Guaranteeing end-to-end timing constraints by calibrating intermediate processes. *IEEE Real-Time Systems Symposium*, 1994.

[46] R. Yerraballi. *Scalability in Real-Time Systems.* PhD thesis, Computer Science Department, old Dominion University, August 1996.

[47] J. Lehoczky and S. Thuel. Algorithms for scheduling hard aperiodic tasks in fixedpriority systems using slack stealing. In *Proceedings of the IEEE Real-Time Systems Symposium*, 1994.

[48] L. Abeni. Server mechanisms for multimedia applications. Technical report, Technical Report RETIS TR98-01, Scuola Superiore S. Anna, 1998.

[49] G. Fohler, T. Lenvall, and G. Buttazzo. Improved handling of soft aperiodic tasks in offline scheduled real-time systems using total bandwidth server. In *8th IEEE Int. Conf. on Emerging Technologies and Factory Automation Nice, France*, October 2001.

[50] M. Nicholson, A. Burns, K. Tindell, and N. Zhang. Allocation of safety critical hard real-time tasks on a parallel processing platform. Technical report, Technical Report YCS-94-238, Department of Computer Science, University of York, 1994.

[51] K. Tindell, A. Burns, and A. Wellings. Allocating hard real-time tasks (an np-hard problem made easy). *Real-Time Systems*, 4(2), 1992.

[52] P. Alternberd. Multiprocessor allocation of periodic hard real-time tasks. Technical report, Technical Report 10/96, C-LAB, C-LAB Furstenallee 11 D 33095 Paderborn, July 1996.

[53] M. Coli and P. Palazzari. A new method for optimisation of allocation and scheduling in real-time applications. In *Proceedings of the 7th Euromicro Workshop on Real-Time Systems*, pages 262–269, 1995.

[54] M. DiNatale and J. A. Stankovic. Applicability of simulated annealing methods to real-time scheduling and jitter control. In *Proceedings of the 7th Euromicro Workshop on Real-Time Systems*, 1995.

[55] M. Nicholson. *Selecting a Topology for Safety-Critical Real-Time Control Systems*. PhD thesis, Department of Computer Science, University of York, September 1998.

[56] Constantine D. Polychronopoulos. *Parallel Programming and Compilers*. Kluwer Academic Publishers, 1988.

[57] V. Sarkar. *Partitioning and Scheduling for Execution on Multiprocessors*. PhD thesis, Department of Computer Science, Stanford University, April 1996.

[58] O. Redell. Modelling and implementing analysis of the distributed real-time control system for a four legged vehicle. Technical report, Technical report, Dept. of Machine design, KTH, TRITA-MMK 1998:8, ISSN 1400-1179, ISRN KTH/MMK–98/8–SE, 1998, 1998.

[59] O. Redell and M. Törngren. Preliminary design of models for the aida tool-set working document. Technical report, Technical report, Dept. of Machine design, KTH, TRITA-MMK 1998:7, ISSN 1400-1179, ISRN KTH/MMK–98/7–SE, 1998, 1998.

[60] E. G. Coffman. *Computer and Job-shop Scheduling Theory*. John Willey & Sons, 1976.

[61] Apostolos Gerasoulis, Sesh Venugopal, and Tao Yang. Clustering task graphs for message passing architectures. In *Proceedings 1990 International Conference on Supercomputing, ACM SIGARCH Computer Architecture News*, volume 18, pages 447–456, 1990.

[62] V. Kianzad and S. Bhattacharyya. Multiprocessor clustering for embedded systems. In *Proceedings of the European Conference on Parallel Computing, Manchester, United Kingdom*, August 2001.

[63] K. Sandström and C. Norström. Managing temporal constraints in control systems. Technical report, Mälardalen Real-Time Research Centre, MRTC Technical report 02/45, 2002.

[64] C. Tudge. *The Engineer in the Garden*. Jonathon Cape, 1993.

[65] C. R. Houck, J. Joines, and M. Kay. Utilising lamarkian evolution and the baldwin effect in hybrid genetic algorithms. Technical report, Technical report, Dept. of Industrial Engineering, North Carolina State University, Raleigh, NC, US, 1995.

[66] L. Davis. *Handbook of Gentic Algorithms.* ISBN 0-442-00173-8. Van Nostrand Reinhold, 1991.

[67] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* ISBN 0-201-15767-5. Addison-Wesley, 1989.

[68] K. Dowsland. Variants of simulated annealing for practical problem solving. In *Proceedings of Adaptive Computing and Information Processing Conference*, pages 115–133, 1994.

[69] V. Kumar and L. N. Kanal. A general branch and bound formulation for understanding and synthesizing and/or tree search procedures. *Artificial Intelligence*, 21:179–198, 1983.

[70] M. Åkerholm and J. Fredriksson. A sample of component technologies for embedded systems. Technical report, Technical report, Mälardalen Real-Time Research Centre, Mälardalen University, November 2004.

[71] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. ISBN 0-201-70332-7. Addison-Wesley, 2001.

[72] M. de Jonge, J. Muskens, and M. Chaudron. Scenario-based prediction of run-time resource consumption in component-based software systems. In *Proceedings of the 6th ICSE Workshop on Component-Based Software Engineering (CBSE6)*, May 2003.

[73] E. Bondarev, J. Muskens, P. de With, M. Chaudron, and J. Lukkien. Predicting real-time properties of component assemblies: a scenario-simulation approach. In *Proceedings of the 30th Euromicro conference, Rennes, France*. IEEE, 2004.

[74] E. Bondarev, J. Muskens, P. de With, and M. Chaudron. Towards predicting real-time properties of a component assembly. In *Proceedings of the 30th Euromicro conference, Rennes, France*. IEEE, 2004.

[75] T. Genssler, A. Christoph, B. Schulz, M. Winter, C. M. Stich, C. Zeidler, P. Müller, A. Stelter, O. Nierstrasz, S. Ducasse, G. Arévalo, R. Wuyts,

P. Liang, B. Schönhage, and R. van den Born. Pecos in a nutshell. Technical report, Technical report, The PECOS Consortium, 2002.

[76] D. B. Stewart, D. E. Schmitz, and P. Khosla. The chimera ii real-time operating system for advanced sensor-based control applications. *IEEE Transactions on Sysytems, Man, and Cybernetics*, 22(6), December 2002.

[77] IEC. International standard IEC 1131: Programmable controllers, 1992.

II

Included Papers

Chapter 6

Paper A: Calculating Resource-Trade-offs when Mapping Components to Real-Time Tasks - extended paper

Johan Fredriksson, Kristian Sandström and Mikael Åkerholm
Extended paper of *Calculating Resource-Trade-offs when Mapping Components to Real-Time Tasks* In the 8th International Symposium on Component-based Software Engineering (CBSE8).

Abstract

The embedded systems domain represents a class of systems that have high requirements on cost efficiency as well as run-time properties such as timeliness and dependability. The research on component-based systems has produced component technologies for guaranteeing real-time properties. However, the issue of saving resources by allocating several components to real-time tasks has gained little focus. Trade-offs when allocating components to tasks are, e.g., CPU-overhead, footprint and integrity. In this paper we present a general approach for allocating components to real-time tasks, while utilizing existing real-time analysis to ensure a feasible allocation. We demonstrate that CPU-overhead and memory consumption can be reduced by as much as 48% and 32% respectively for industrially representative systems.

6.1 Introduction

Many real-time systems (RTS) have high requirements on safety, reliability and availability. Furthermore the development of embedded systems is often sensitive to system resource usage in terms of, e.g., memory consumption and processing power. Historically, to guarantee full control over the system behaviour, the development of embedded systems has been done using only low level programming. However, as the complexity and the amount of functionality implemented by software increase, so does the cost for software development. Also, since product lines are common within the domain, issues of commonality and reuse are central for reducing cost. Component-Based Development (CBD) has shown to be an efficient and promising approach for software development, enabling well defined software architectures as well as reuse. Hence, CBD can be used to achieve goals such as cost reduction, and quality and reliability improvements.

In embedded RTS timing is important, and scheduling is used to create predictable timing. Furthermore, these systems are often resource constrained; consequently memory consumption and CPU load are desired to be low. A problem in current component-based embedded software development practices is the allocation of components to run-time tasks [1]. Because of the real-time requirements on most embedded systems, it is vital that the allocation considers temporal attributes, such as worst case execution time (WCET), deadline (D) and period time (T). Hence, to facilitate scheduling, components are often allocated to tasks in a one-to-one fashion. However, for many embedded systems it is desired to optimize for memory and speed [2], thus the one-to-one allocation is unnecessarily memory and CPU consuming.

Embedded RTS consist of periodic and sporadic events that usually have end-to-end timing requirements. Components triggered by the same periodic event can often be coordinated and executed by the same task, while still preserving temporal constraints. Thus, it is easy to understand that there can be profits from allocating several components into one task. Some of the benefits are less memory consumption in terms of stacks and task control blocks or lower CPU utilization due to less overhead for context switches. Different properties can be accentuated depending on how components are allocated to tasks, e.g., memory usage and performance; Hence, there are many trade-offs to be made when allocating components to tasks.

Allocating components to tasks, and scheduling tasks are both complex problems and different approaches are used. Simulated annealing and genetic algorithms are examples of algorithms that are frequently used for optimization

problems. However, to be able to use such algorithms, a framework to calculate properties, such as memory consumption and CPU-overhead, is needed. The work presented in this paper describes a general framework for reasoning about trade-offs concerning allocating components to tasks, while preserving extra-functional requirements. Temporal constraints are verified and the allocations are optimized for low memory consumption and CPU-overhead. The framework is evaluated using industrially relevant component assemblies, and the results show that CPU-overhead and memory consumption can be reduced by as much as 48% and 32% respectively.

The idea of assigning components to tasks for embedded systems while considering extra-functional properties and resource utilization is a relatively uncovered area. In [3, 4] Bondarev et. al. are looking at predicting and simulating real-time properties on component assemblies. However, there is no focus on increasing resource utilization through component to task allocation. The problem of allocating tasks to different nodes is a problem that has been studied by researchers using different methods [5, 6]. There are also methods proposed for transforming structural models to run-time models [7, 8, 1], but extra-functional properties are usually ignored or considered as non-critical [9]. In [10], an architecture for embedded systems is proposed, and it is identified that components has to be allocated to tasks, however there is no focus on the allocation of components to tasks. In [9] the authors propose a model transformation where all components with the same priority are allocated to the same task; however no consideration is taken to lower resource usage. In [11], the authors discuss how to minimize memory consumption in real-time task sets, though it is not in the context of allocating components to tasks. Shin et. al [12] are discussing the code size, and how it can be minimized, but does not regard scheduling and resource constraints.

The outline for the rest of the paper is as follows; section 2 gives an overview of the component to task allocations, and describes the structure of the components and tasks. Section 3 describes a framework for calculating the properties of components allocated to tasks. Section 4 discusses allocation and scheduling approaches, while evaluations and simulations are presented in section 5. Finally in section 6, future work is discussed and the paper is concluded. Detailed data regarding the simulations can be found in appendix A.

6.2 Allocating components to real-time tasks

In RTS temporal constraints are of great importance and tasks control the execution of software. Hence, components need to be allocated to tasks in such a way that temporal requirements are met, and resource usage is minimized. Given an allocation we determine if it is feasible and calculate the memory consumption and task switch overhead. To impose timing constraints, we define end-to-end timing requirements and denote them transactions. Transactions are defined by a sequence of components and a deadline. Thus, the work in this paper has three main concerns:

1. Verification of allocations from components to tasks.
2. Calculating system properties for an allocation
3. Minimizing resource utilization

CBSE is generally not used when developing embedded RTS, mostly due to the lack of efficient mappings to run-time systems and real-time properties. One approach that allows an efficient mapping from components to a RTS is the Autocomp technology [13]. An overview of the Autocomp technology can be seen in Figure 6.1. The different steps in the figure are divided into design-time, compile-time, and run-time to display at which point in time during development they are addressed or used. The compile-time steps, illustrated in Figure 6.1, incorporate an allocation from the component-based design, to a real-time model and mapping to a real-time operating system (RTOS). During this step the components are allocated to real-time tasks and the component requirements are mapped to task-level attributes.

By combining the notion of transactions and the pipe-and-filter interaction model we get a general component model that is easy to implement for a large set of component technologies for embedded systems such as Autocomp [13], SaveCCM [14], Rubus [15], Koala [16], Port-based objects [17], IEC61131[18] and Simulink[19]. The component model characteristics are described in the section 6.2.1 and the task model characteristics are described in section 6.2.2.

6.2.1 Component model characteristics

In this section we describe characteristics for a general component model that is applicable to a large set of embedded component models. Both component and task models described are meta-models for modelling the most important

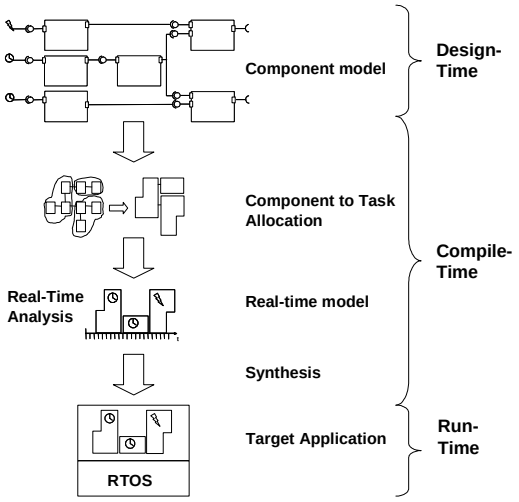

Figure 6.1: Autocomp system description

attributes of an allocation between components and tasks. The component interaction model used throughout this paper is a pipe-and-filter model with transactions. Each component has a trigger; a time trigger or an event trigger or a trigger from a preceding component. A component transaction describes an order of components and defines an end-to-end timing requirement. In Figure 6.2, the notation of a component assembly with six components and four transactions is described. The graphical notation is similar to the one used in UML.

The component model chosen is relatively straight forward to analyse and verify. The pipe-and-filter interaction model is commonly used within the embedded systems domain. Many component models for embedded systems have the notion of transactions built in; however, if a component model lacks the notion of transactions, there are often possibilities to model end-to-end timing requirements and execution order at a higher abstraction level. In general a system is described with components, component relations, and transactions (flow) between components. The component model is described with:

Component c_i is described with the tuple $< S_i, Q_i, X_i, M_i >$, where S_i is a signal from another component, an external event or a timed event.

Q_i represents the *minimum inter arrival time* (MINT) in the case of an external event. It represents the period in the case of a timed trigger and it is unused if the signal is from another component. The parameter X_i is the WCET for the component, and M_i is the amount of stack required by the component.

Isolation set I defines a relation between components that should not be allocated. It is described with a set of component pairs $I =< (c_1, c_2), (c_3, c_4) >$ that define what components may not be allocated to the same task. There may be memory protection requirements or other legitimate engineering reasons to avoid allocating certain combinations of components; for example, if a component has a highly uncertain WCET. The isolation set is indexed with subscripts denoting next inner element, i.e., $I_1 = (c_1, c_2)$ and $I_{12} = c_2$.

Component Transaction ctr_i is an ordered relation between components $N_i = c_1, c_2, ..., cn$, and an end-to-end deadline dc_i. The deadline is relative to the event that triggered the component transaction, and the first component within a transaction defines the transaction trigger. A component transaction can stretch over one or several components, and a component can participate in several component transactions. The component c_a should execute before the component c_b, and the component c_b should execute before c_c to produce the expected results etc. The correct execution behaviour for the set $N = c_1, c_2, ..., c_n$ can be formalized with the regular expression denoted in 6.1.

$$c_1 \Sigma^* c_2 \Sigma^* ... c_n \tag{6.1}$$

Where Σ^* denotes all allowed elements defined by N.

In a component assembly, event triggers are treated different from the periodic triggers as the former is not strictly periodic. There is only a lower boundary restricting how often it can occur, but there is no upper bound restricting how much time may elapse between two invocations. Thus, if an event trigger could exist inside or last in a transaction, it would be impossible to calculate the response time for the transaction, and hence a deadline could never be guaranteed.

6.2.2 Task characteristics

The task model specifies the organization of entities in the component model into tasks and transactions over tasks. During the transformation from component model to run-time model, extra-functional properties like schedulability

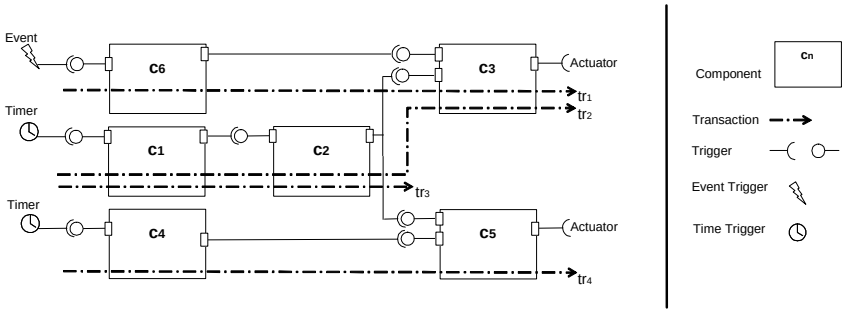

Figure 6.2: Graphical notation of the component model.

and response-time constraints must be considered in order to ensure the correctness of the final system. Components only interact through explicit interfaces; hence tasks do not synchronize outside the component model. The task model is for evaluating schedulability and other properties of a system, and is similar to standard task graphs as used in scheduling theory, augmented with exclusion constraints (isolation). The task model is described with:

System K is described with the tuple $< A, \tau, \rho >$ where A is a task set scheduled by the system. The constant τ is the size of each task control block, and can be considered constant and the same for all tasks. The constant ρ is the time associated with a task switch. The system kernel is the only explicitly shared resource between tasks; hence we do not consider blocking. Also blocking is not the focus of this paper.

Task t_i is described with the tuple $< C_i, T_i, wcet_i, stack_i >$ where C_i is an ordered set of components. Components within a task are executed in sequence. Components within a task are executed at the same priority as the task, and a high priority task pre-empts a low priority task. T_i is the period or minimum inter arrival time of the task. The parameters $wcet_i$ and $stack_i$ are worst case execution time and stack size respectively. The $wcet_i$, $stack_i$ and period (T_i) are deduced from the components in C_i. The $wcet_i$ is the sum of all the WCETs for all components allocated to the task. Hence, for a task t_i, the parameters $wcet_i$ and $stack_i$ are

calculated with (2) and (3) .

$$wcet_n = \sum_{\forall_i (c_i \in C_n)} (X_i) \tag{6.2}$$

$$stack_n = \forall_i (c_i \in C_n) max(M_i) \tag{6.3}$$

Task transaction ttr_i is a sequence of tasks $O_i = t_1, t_2, ..., t_k$ and a relative deadline dt_i. O_i defines an ordered relation between the tasks, where in the case of $O = t_1, t_2$; t_1 is predecessor to t_2. The timing and execution order requirements of a task transaction ttr_i are deduced from the requirements of the component transactions ctr_i. The task transaction ttr_i has the same parameter as the component transactions ctr_i but t_1, t_2,..., t_k are the tasks that map the component $c_a, c_b, ..., c_n$, as denoted in Figure 6.4. If several task transactions ttr_i span over the exact same tasks, the transactions are merged and assigned the shortest deadline. An event-triggered task may only appear first in a transaction. Two tasks can execute in an order not defined by the transactions. This depends on that the tasks have different period times, and thereby suffer from period phasing; hence transactions can not define a strict precedence relation between two tasks. Figure 6.3 is an execution trace that shows the relation between tasks and transactions. The tasks and transactions are the same as in Figure 6.4, left part.

6.3 Allocation framework

The allocation framework is a set of models for calculating properties of allocations of components to tasks. The properties calculated with the framework are used for optimization algorithms to find feasible allocations that fulfil given requirements on memory consumption and CPU-overhead.

For a task set A that has been mapped from components in a one-to-one fashion, it is trivial to calculate the system memory consumption and CPU-overhead since each task has the same properties as the basic component. When several components are allocated to one task we need to calculate the appropriateness of the allocation and the tasks properties. For a set of components, c_1,...,c_n, allocated to a set of tasks A, the following properties are considered.

- CPU-overhead p_A

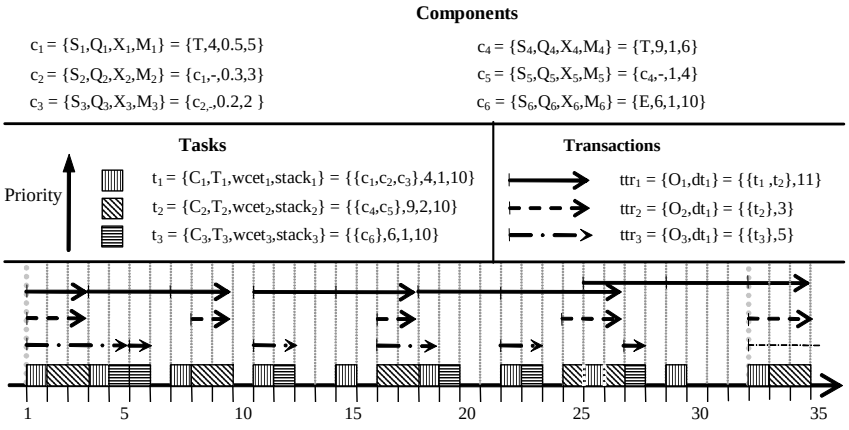

Figure 6.3: Task execution order and task transactions.

- Memory consumption m_A

Each component c_i has a memory consumption stack. The stack of the task is the maximum size of all components stacks allocated to the task since all components will use the same stack. The CPU overhead p, the memory consumption m for a task set A in a system K are formalized in equations 6.4 and 6.5:

$$p_A = \sum_{\forall_i (t_i \in A)} \frac{\rho}{T_i} \tag{6.4}$$

$$m_A = \sum_{\forall_i (t_i \in A)} (stack_i + \tau) \tag{6.5}$$

Where p_A represents the sum of the task switch overhead divided by the period for all tasks is the system, and m_A represents the total amount of memory used for stacks and task control blocks for all tasks in the system

6.3.1 Constraints on allocations

There is a set of constraints that must be considered when allocating components. These are:

- Component isolation

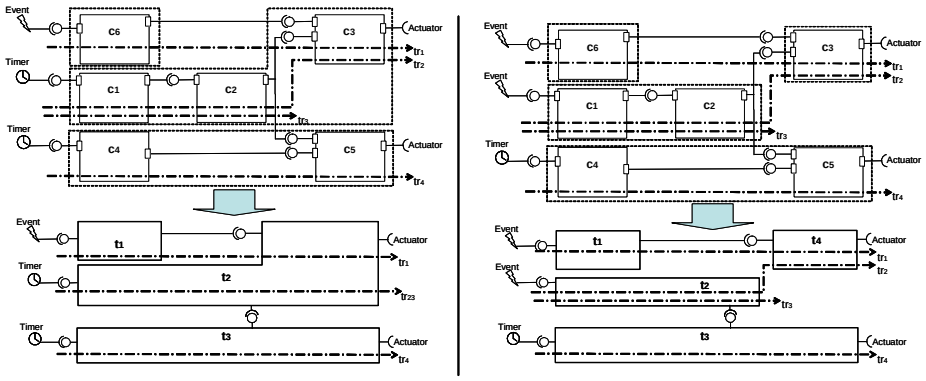

Figure 6.4: Two allocations from components to tasks dependent on intersecting transactions.

- Intersecting transactions
- Trigger types and period times
- Schedulability

Each constraint is further discussed below:

Isolation

It is not realistic to expect that components can be allocated in an arbitrary way. There may be explicit dependencies that prohibits that certain components are allocated together, therefore the isolation set I defines which components may not be allocated together. There may be specific engineering reasons to why some components should be separated. For instance, it may be desired to minimize the jitter for some tasks, thus components with highly uncertain WCET should be isolated. There may also be integrity reasons to separate certain combinations of components. Hence it must be assured that two components that are defined to be isolated do not reside in the same task. This can be validated with equation 6.6:

$$\begin{aligned} &Iso(a,b) : c_a \text{ has an isolation requirement to } c_b \\ &\neg\exists_i(\forall_j\forall_k(c_j \in C_i \wedge c_k \in C_i \wedge Iso(j,k))) \end{aligned} \tag{6.6}$$

Where there must not exist any task t_i that has two components c_j and c_k, if these components have an isolation requirement.

Intersecting transactions

If component transactions intersect, there are different strategies for how to allocate the component where the transactions intersect. The feasibility is described in equations 6.7 and 6.8. A component in the intersection should not be allocated with any preceding component if both transactions are event triggered; the task should be triggered by both transactions to avoid pessimistic scheduling. A component in the intersection of one time-triggered transaction and one event-triggered transaction can be allocated to a separate task, or with a preceding task in the time-triggered transaction. A component in the intersection of two time-triggered transactions can be allocated arbitrarily. In Figure 6.4, two different allocations are imposed due to intersecting event-triggered transactions. In the left part of Figure 6.4 there is an intersection between a time triggered and an event triggered transaction. Then the intersecting component c_3 is allocated to the task triggered by the time triggered transaction. In the right part of the figure, where two event triggered transactions intersect, the component c_3 is allocated to a separate task, triggered by both transactions.

$$T_E(tr) : \text{transaction is event triggered}$$
$$T_T(tr) : \text{transaction is time triggered}$$
$$P(a,b,d) : c_a \text{ is predecessor to } c_b \text{ in the set } N_d$$
$$X_a^{bc} = c_a \in N_b \wedge c_a \in N_c$$
$$Y_{ab}^{c} = c_a \in C_c \wedge c_b \in C_c$$

$$\neg\exists_i(\forall_j\forall_k\forall_l\forall_m(X_l^{jk} \wedge Y_{lm}^{i} \wedge T_E(ctr_j) \wedge T_E(ctr_k) \wedge (P(m,l,k) \vee P(m,l,j)))) \quad (6.7)$$

$$\neg\exists_i(\forall_j\forall_k\forall_l\forall_m(X_l^{jk} \wedge Y_{lm}^{i} \wedge c_m \in N_k \wedge T_T(ctr_j) \wedge T_E(ctr_k) \wedge P(c_m, c_l, N_k))) \quad (6.8)$$

Where there must not exist any task t_i that has two components c_l and c_m in a way that two component transactions ctr_j and ctr_k intersect in c_l, and c_m precedes c_l in the transactions ctr_j or ctr_k, if ctr_j or ctr_k are event-triggered.

Triggers

Some allocations from components to tasks can be performed without impacting the schedulability negatively. A component that triggers a subsequent component can be allocated into a task if it has no other explicit dependencies, see

(1) in Figure 6.5. Components with the same period time can be allocated together if they do not have any other explicit dependencies, see (2) in Figure 6.5. To facilitate analysis, a task may only have one trigger, so time triggered components with the same period can be triggered by the same trigger and thus allocated to the same task. However, event triggered components may only be allocated to the same task if they in fact trigger on the same event, and have the same minimum inter arrival time, see (3) in Figure 6.5. Components with harmonic periods could also be allocated to the same task. However, harmonic periods create jitter. Consider two components with the harmonic periods five and ten that are allocated to one task. The component with the period five will run every invocation, while the other component will run every second invocation, which creates a jitter; therefore we have chosen not to pursue this specific issue.

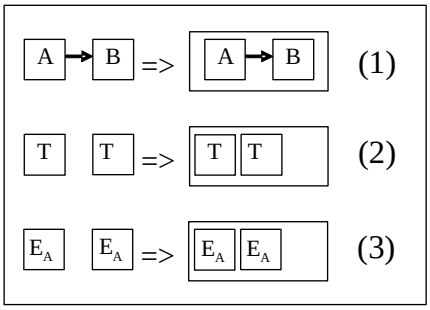

Figure 6.5: Component to task allocation considering triggers.

Schedulability

Schedulability analysis is highly dependent on the scheduling policy chosen. Depending on the system design, different analyses approaches have to be considered. The task and task transaction meta-models are constructed to fit different scheduling analyses. In this work we have used fixed priority exact analysis. However, the model can easily be extended with jitter and blocking for real-time analysis models that use those properties. The framework assigns each task a unique priority pre run-time, and it uses exact analysis for schedulability analysis, together with the Bate and Burns [20] approach for verifying that the transaction deadlines are met.

6.4 Using the framework

An allocation can be performed in several different ways. In a small system all possible allocations can be evaluated and the best chosen. For a larger system, however, this is not possible due to the combinatorial explosion. Different algorithms can be used to find a feasible allocation and scheduling of tasks. For any algorithm to work there must be some way to evaluate an allocation or real-time schedule. The proposed allocation framework can be used to calculate schedulability, CPU-overhead and total memory load. The worst-case allocation is a one-to-one allocation where every component is allocated to one task. The best-case allocation on the other hand, is where all components are allocated to one single task. To allocate all components to one task is very seldom feasible. Also, excessive allocation of components may negatively affect scheduling, because the granularity is coarsened and thereby the flexibility for the scheduler is reduced.

Simulated annealing, genetic algorithms and bin packing are well known algorithms often used for optimization problems. These algorithms have been used for problems similar to those described in this paper; bin packing, e.g., has been proposed in [21] for real-time scheduling. Here we briefly discuss how theses algorithms can be used with the described framework, to perform component to task allocations.

Bin Packing is a method well suited for our framework. In [22] a bin packing model that handles arbitrary conflicts (BPAC) is presented. The BPAC model constrains certain elements from being packed into the same bin, which directly can be used in our model as the isolation set I. The bin-packing feasibility function is the schedulability, and the CPU and memory overhead constitute the optimization function.

Genetic algorithms can solve, roughly, any problem as long as there is some way of comparing two solutions. The framework proposed in this paper give the possibility to use the properties memory consumption, CPU-overhead and schedulability as grades for an allocation, in order to evolve new allocation specimen. In, e.g., [23] and [24], genetic algorithms are used for scheduling complex task sets and scheduling task sets in distributed systems.

Simulated annealing (SA) is a global optimization technique that is regularly used for solving NP-Hard problems. The energy function consists of a schedulability test, the memory consumption and CPU-overhead. In

[6] and [25] simulated annealing is used to place tasks on nodes in a distributed system.

6.5 Evaluation

In order to evaluate the performance of the allocation approach the framework has been implemented. We have chosen to perform a set of allocations and compare the results to a corresponding one-to-one allocation where each component is allocated to a task. We compare the allocations with respect to if the allocation is feasible (real-time analysis), memory consumption and CPU overhead. The implementation is based on genetic algorithms (GA) [26], and as Figure 6.6 shows, each gene represents a component and contains a reference to the task it is assigned. Each chromosome represents the entire system with all components assigned to tasks. Each allocation produced by the GA is evaluated by the framework, and is given a fitness value dependent on the validity of the allocation, the memory consumption and the CPU overhead.

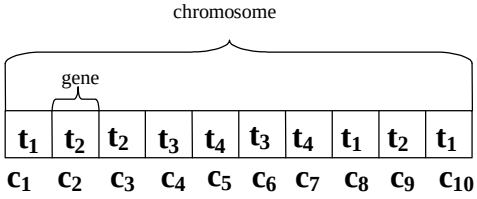

Figure 6.6: The genetic algorithm view of the component to task allocation; A system with ten components, allocated to four tasks.

6.5.1 Fitness function

The fitness function is based on the feasibility of the allocation together with the memory consumption and CPU overhead. The feasibility part of the fitness function is mandatory, i.e., the fitness value for a low memory and CPU overhead can never exceed the value for a feasible allocation. The feasibility function consists of: I which represents component isolation, IT representing intersecting transactions, Tr representing trigger types and period times, and finally Sc represents scheduling. Consider that each of these feasibility tests

are assigned a value greater than 1 if they are true, and a value of 0 if they are false. The parameter n represents the total number of components. Then, the fitness function can be described as with equation 6.9.

$$Fitness = \left((I{+}IT{+}Tr{+}Sc)F{+}\Big(\frac{n}{m_A}{+}\sum_{\forall i(t_i \in A)} \frac{\rho \cdot n}{T_i}\Big)O \right) \cdot (I{\cdot}IT{\cdot}Tr{\cdot}Sc{+}1) \tag{6.9}$$

Where the fitness is the sum of all feasibility values times a factor F, added with the inverted memory usage and performance overhead, times a factor O. The total fitness is multiplies with 1 if any feasibility test fail, and the products of all feasibility values plus 1 if all feasibility tests succeed.

6.5.2 Simulation set up

This section describes the simulation method and set up. For each simulation the genetic algorithm assigns components to tasks and evaluates the allocation, and incrementally finds new allocations. The evaluation is performed in a number steps:

1. System data - components and transactions with deadlines are created. There exist at least one solution for all data that are passed on to the GA.
2. Initial Population - The GA creates a random population that makes up a set of allocations. One population comprises several chromosomes, and each chromosome represents an allocation.
3. Apply Fitness function - The fitness function calculates how fit a chromosome is. The higher fitness value, the more likely is the chromosome to be passed on to the next generation.
4. Create New population - The GA combines different chromosomes, and performs mutations by reassigning one or several components.
5. Repeat from step 3, each iteration is referred to as a generation.

The system data is produced by creating a random schedulable task set, on which all components are randomly allocated. The component properties are deduced from the task they are allocated. Transactions are deduced the same way from the task set. In this way it is always at least one solution for each system. However, it is not sure that all systems are solvable with a one-to-one

allocation. The components and component transactions are used as input to the framework. Hereafter, systems that are referred to as generated systems are generated to form input to the framework. Systems that come out of the framework are referred to as allocated systems. The simulation parameters are set up as follows:

- The number of components of a system is randomly selected from a number of predefined sets. The numbers of components in the systems are ranging in twenty steps from 40 to 400, with a main point on 120 components.
- The period times for the components are randomly selected from a predefined set of different periods.
- The worst case execution time (WCET) is specified as a percentage of the period time and chosen from a predefined set. The WCETs together with the periods in the system constitutes the system load.
- The transaction size is the size of the generated transactions in percentage of the number of components in the system. The transaction size is randomly chosen from a predefined set. The longer the transactions, the more constraints on how components may be allocated.
- The transaction deadline laxity is the percentage of the lowest possible transaction deadline for the generated system. The transaction deadline laxity is evenly distributed among all generated systems and is always greater or equal to one, to guarantee that the generated system is possible to map. The higher the laxity, the less constrained transaction deadlines.

One component can be involved in more than one transaction, resulting in more constraints in terms of timing. The probability that a component is participating in two transactions is set to 50% for all systems.

To get as realistic systems to simulate as possible, the values used to generate systems are gathered from some of our industrial partners. The industrial partners chosen are active within the vehicular embedded system segment. A complete table with all values and distributions, of the system generation values, can be found in appendix A. The task switch time used for the system is 22 μs, and the tcb size is 300 bytes. The task switch time and tcb size are representative of commercial RTOS tcb sizes and context switch times for common CPUs.

The simulations are performed for four different utilization levels, 30%, 50%, 70% and 90%. For each level of utilization 1000 different systems are generated with the parameters presented above.

6.5.3 Results

A series of simulations have been carried out to evaluate the performance of the proposed framework. To evaluate the schedulability of the systems, FPS scheduling analysis is used. The priorities are randomly assigned by the genetic algorithm, and no two tasks have the same priority. The simulations compare the approach in this paper to a one-to-one allocation. Table 6.5.3 summarizes the results from the simulations. The columns entitled "stack" and "CPU" displays the average memory size (stack + tcb) and CPU overhead respectively, for all systems with a specific load and transaction deadline laxity. The column entitled "success" in the 1-1 allocation section displays the rate of systems that are solvable with the 1-1 allocation. The column entitled "success" in the GA allocation section displays the rate at which our framework finds allocations, since all systems has at least one solution. The stack and CPU values are only collected from systems where a solution was found.

The first graph for the simulations (Figure 6.7) shows the success ratio, i.e., the percentage of systems that were possible to map with the one-to-one allocation, and the GA allocation respectively. The success ratio is relative to the effort of the GA, and is expected to increase with a higher number of generations for each system. Something that might seem confusing is that the success ratio is lower for low utilization than for high utilizations, event though it, intuitively, should be the opposite. The explanation to this phenomenon is that the timing constraints become tighter as fewer tasks participate in each transaction (lower utilization often leads to fewer tasks). With fewer tasks the task phasing, due o different periods, will be lower, and the deadline can be set tighter.

The second graph (Figure 6.8) shows that the deadlines are relaxed with higher utilization, since the allocations with relaxed deadlines perform well, and the systems with a more constrained deadline show a clear improvement with higher utilization.

The third graph (Figure 6.9) shows for both approaches the average stack size for the systems at different utilization. The comparison is only amongst allocations that are have been mapped by both strategies. The memory size is consistent of the tcb and the stack size. The tcb size is 300 byte. As described earlier, each task allocates a stack that is equal to the size of the largest stack

Load	Laxity	1-1 allocation			GA allocation		
		Stack	CPU	success	stack	CPU	success
	All	28882	4,1%	74%	17380	2,0%	87%
30%	1.1	25949	3,5%	39%	14970	1,6%	58%
	1.3	33077	4,4%	78%	21005	2,2%	97%
	1.5	26755	4,1%	95%	15503	2,0%	99%
50%	All	37277	4,8%	82%	24297	2,4%	90%
	1.1	35391	4,3%	49%	23146	2,3%	64%
	1.3	38251	4,8%	88%	25350	2,5%	96%
	1.5	37043	4,9%	98%	23740	2,3%	100%
70%	All	44455	5,1%	85%	30694	2,7%	91%
	1.1	44226	5,0%	58%	31638	2,7%	73%
	1.3	44267	5,1%	94%	30686	2,7%	98%
	1.5	44619	5,2%	98%	30232	2,6%	100%
90%	All	46943	5,6%	87%	37733	3,1%	93%
	1.1	54858	5,7%	65%	41207	3,4%	80%
	1.3	49607	5,5%	92%	35470	3,0%	98%
	1.5	53535	5,7%	98%	38260	3,1%	99%

Table 6.1: Memory, CPU overhead and success ratio for 1-1 and GA allocations

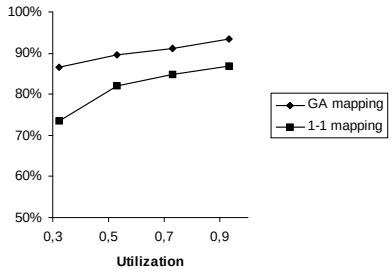

Figure 6.7: Average success ratio

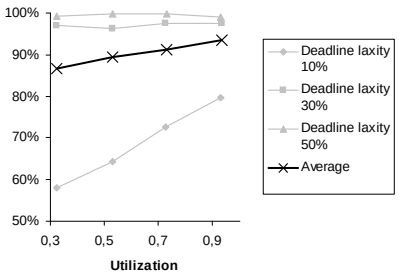

Figure 6.8: Success rate for allocations

among its allocated components.

The fourth graph (Figure 6.10) shows the average task switch time in micro seconds for the entire system. The task switch overhead is only dependent on how many tasks there are in the system. The average improvement of GA allocation in comparison to the 1-1 allocation is, for the success ratio, 10%. The memory size is reduced by 32%, and the task switch overhead is reduced by 48%. Hence we can see a substantial improvement in using smart methods to map components to tasks. A better strategy for setting priorities would probably lead to an improvement in the success ratio. This is expected because the constraints are more relaxed, allowing for more freedom in the allocation. Further we see that lower utilization give better improvements than higher laxity of the deadlines. Since lower utilization often in the simulations give tighter deadlines, we can conclude that the allocation does not negatively impact schedulability. However, the more components, i.e., the higher load, the more constrains are put on the transactions, and thereby on the components, making it harder to perform a good allocation.

6.6 Conclusions and Future Work

Resource efficiency is important for RTS, both regarding performance and memory. Schedulability, considering resource efficiency, has gained much focus, however the allocation between components to tasks has gained very little focus. Hence, in this paper we have described an allocation framework for allocating components to tasks, to facilitate existing scheduling and optimization algorithms such as genetic algorithms, bin packing and simulated annealing. The framework is designed to be used during compile-time to minimize re-

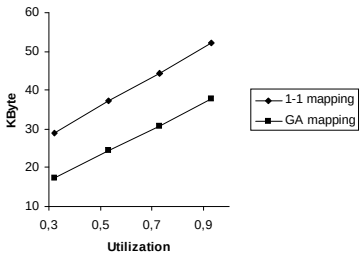

Figure 6.9: Average memory size

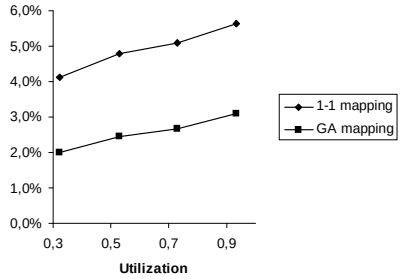

Figure 6.10: Average task switch overhead

source usage and maximize timeliness. It can also be used iteratively in case of design changes; however with some obvious drawbacks on the results. The framework can easily be extended to support other optimizations, besides task switch overhead and memory consumption. Results from simulations show that the framework gives substantial improvements both in terms of memory consumption and task switch overhead. The described framework also has a high ratio in finding feasible allocations. Moreover, in comparison to allocations performed with a one-to-one allocation our framework performs very well, with 32% reduced memory size and 48% reduced task switch overhead. The simulations show that the proposed framework performs allocations on systems of a size that covers many embedded systems, and in a reasonable time for an off-line tool. We have also shown how CPU load and deadline laxity affects the allocation. Future work includes adding other allocation criteria, e.g., by looking at jitter requirements, and blocking. By adding jitter constraints and blocking, trade-offs arise between switch overhead and memory size versus deviation from nominal start and end times and blocking times. Furthermore, a more efficient scheduling policy and priority assignment will be applied. Due to the nature of GA it is easy to add new optimizations as the ones suggested above.

Bibliography

[1] K. Mills. and H. Gomaa. Knowledge-based automation of a design method for concurrent systems. *IEEE Transactions on Software Engineering*, 28(3), 2002.

[2] I. Crnkovic. Component-based approach for embedded systems. In *Ninth International Workshop on Component-Oriented Programming, Oslo*, June 2004.

[3] E. Bondarev, J. Muskens, P. de With, M. Chaudron, and J. Lukkien. Predicting real-time properties of component assemblies: a scenario-simulation approach. In *Proceedings of the 30th Euromicro conference, Rennes, France*. IEEE, 2004.

[4] E. Bondarev, J. Muskens, P. de With, and M. Chaudron. Towards predicting real-time properties of a component assembly. In *Proceedings of the 30th Euromicro conference, Rennes, France*. IEEE, 2004.

[5] C. Hou. and K. G. Shin. Allocation of periodic task modules with precedence and deadline constraints in distributed real-time system. *IEEE Transactions on Computers*, 46(12), 1995.

[6] K. Tindell, A. Burns, and A. Wellings. Allocating hard real-time tasks (an np-hard problem made easy). *Real-Time Systems*, 4(2), 1992.

[7] B. P. Douglas. *Doing Hard Time*. 0201498375. Addison Wesely, 1999.

[8] H. Gomaa. *Designing Concurrent Distributed, and Real-Time Applications with UML*. 0-201-65793-7. Addison Wesely, 2000.

[9] S. Kodase, S. Wang, and K. G. Shin. Transforming structural model to runtime model of embedded software with real-time constraints. In *In*

proceeding of Design, Automation and Test in Europe Conference and Exhibition, pages 170–175. IEEE, 1995.

[10] K. G. Shin and S. Wang. An architecture for embedded software integration using reusable components. In *proceeding of the international conference on Compilers, architectures, and synthesis for embedded systems, San Jose, California, United States*, pages 110–118. IEEE, 2000.

[11] P. Gai, G. Lippari, and M. Di Natale. Minimizing memory utilization of real-time task sets in single and multi-processor systems-on-a-chip. In *Proceedings of the Real-Time Systems Symposium, London (UK) Dec*. IEEE, 2001.

[12] K. G. Shin, I. Lee, and M. Sang. Embedded system design framework for minimizing code size and guaranteeing real-time requirements. In *Proceedings of the 23rd IEEE Real-Time Systems Symposium, RTSS 2002, Austin, TX, December 2-5*. IEEE, 2002.

[13] K. Sandstrom, J. Fredriksson, and M. Åkerholm. Introducing a component technology for safety critical embedded real-time systems. In *Proceeding of CBSE7 International Symposium on Component-based Software Engi-neering*. IEEE, 2004.

[14] H. Hansson, M. Åkerholm, I. Crnkovic, and M. Törngren. Saveccm - a component model for safety-critical real-time systems. In *Euromicro Conference, Special Session Component Models for Dependable Systems Rennes, France*. EEE, September 2004.

[15] Arcticus. Arcticus homepage: http://www.arcticus.se.

[16] R. van Ommering, F. van der Linden, and J. Kramer. The koala component model for consumer electronics software. In *IEEE Computer*, pages 78–85. IEEE, March 2000.

[17] D. B. Stewart, R. A. Volpe, and P. K. Khosla. Design of dynamically reconfigurable real-time software using port-based objects. In *IEEE Transactions on Software Engineering*, pages 759–776. IEEE, December 1997.

[18] IEC. International standard IEC 1131: Programmable controllers, 1992.

[19] Mathworks. Mathworks homepage : http://www.mathworks.com.

[20] A. Bate and I. Burns. An approach to task attribute assignment for uniprocessor systems. In *Proceedings of the 11th Euromicro Workshop on Real Time Systems, York, England*. IEEE, June 1999.

[21] Y. Oh and S. H. Son. On constrained bin-packing problem. Technical report, Technical Report, CS-95-14, Univeristy of Virginia, 1995.

[22] K. Jansen and Ohring S. R. Approximation algorithms for time constrained scheduling. In *proceeding of Workshop on Parallel Algorithms and Irregularly Structured Problems*, pages 143–157. IEEE, 1995.

[23] Y. Monnier, J.-P. Beauvis, and J.-M. Deplanche. A genetic algorithm for scheduling tasks in a real-time distributed system. In *Proceeding of 24th Euromicro Conference*, pages 708–714. IEEE, 1998.

[24] D. Montana, M. Brinn, S. Moore, and G. Bidwell. Genetic algorithms for complex, real-time scheduling. In *Proceeding of IEEE International Conference on Systems, Man, and Cybernetics*, pages 2213–2218. IEEE, 1998.

[25] S. T. Cheng. and Agrawala A. K. Allocation and scheduling of real-time periodic tasks with relative timing constraints. In *Second International Workshop on Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 1995.

[26] C. M. Fonseca. and P. J. Flemming. An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary computation*, 3(1), 1995.

Appendix A

In this appendix we show the specific data used for generating systems to the simulations. The software is implemented using java, and for the basic structures and operations of the genetic algorithm the open gnu library Gajit, written by Mathew Faupel. The GA was setup with an initial population of 300 individuals, and every simulation was run for 500 generations. The simulations were run on a 1.8 GHz Pentium 4m processor with 768 MB of RAM. The mean time for each simulation is 133 seconds. The parameters used for the GA in the experiments are shown in table 6.

Param on component level	
Number components	Dist. %
40	1,25
50	6,25
60	10
70	6,25
80	2,75
100	7,5
120	13
140	7,5
150	5
160	2,5
180	8
200	5,25
210	5
240	9
250	1,25
280	5
300	2
320	1
350	1,25
400	0,25
Isolation %	Dist. %
0	20
10	30
20	30
30	20
Period time (ţs)	Dist. %
10000	20
25000	20
50000	40
100000	20

WCET in % of period	Dist. %
2	45
4	50
8	5
Stack size	Dist. %
256	10
512	25
1024	25
2048	35
4096	10
Param on System level	
ctr. size % of num. comp.	Dist. %
10	10
13	25
17	25
21	25
25	15
Laxity % of ctr.dl	Dist. %
110	33
130	33
150	33
Utilization %	Dist. %
30	25
50	25
70	25
90	25
GA parameters	
GA property	Value
Population	300
Generations	500
Elite rate	5%
Cull rate	40%
Mutation rate	1%

Table 6.2: Data used for generating systems, and GA parameter

Chapter 7

Paper B: Attaining Flexible Real-Time Systems by Bringing Together Component Technologies and Real-Time Systems Theory

Johan Fredriksson, Mikael Åkerholm, Radu Dobrin and Kristian Sandström
In Euromicro Conference, Component-Based Software Engineering Track, Belek, Turkey, Septembed 2003

Abstract

The use of real-time systems (RTS) has gained a wide acceptance in a number of industrial applications during the past years, as well as the field is continuously expanding. At the same time, the main challenge for software developers today, is to deal with complexity and to quickly adapt to changes. Component-based software engineering (CBSE) has arised as a technology to meet a number of issues like software reusability, reliability, and a short time to market for industrial products. However, the general advantages of utilizing CBSE techniques for RTS are desirable only if the correct timing can be maintained. Therefore we need real-time theory, which provides run-time mechanisms and analysis to guarantee the timeliness in the system based on an upper bound of the execution time. In this paper we present a component model, together with run-time mechanisms, gathering benefits provided by both RTS and CBSE. In particular, we show that the proposed model is a suitable package for efficient utilization of the multiple version paradigm. The purpose for using a multiple version technique is to ensure real-time guarantee for a minimum level of service quality while providing user-specified run-time flexibility in terms of increased level of quality based on resource availability.

7.1 Introduction

Computer control systems are embedded in a large and growing group of products. Products such as automotive vehicles, aircraft, and industrial robots are equipped with advanced computer control systems and have high requirements of reliable and safe operation. A motivation for establishing a CBSE discipline for such systems is that the systems are becoming increasingly more complex due to the inclusion of more functionality. At the same time, the product cycles are becoming shorter leading to requirements of shorter time to market. Moreover, the industry strives to enable cost effective implementation of new functionality. Thus, the challenge is cost efficient development of these systems, with respect to business, quality, reliability, and functionality.

Although the motivation for utilizing CBSE methods for development of embedded control systems are essentially the same as for general purpose software, the requirements on a component model for embedded systems are not the same. For embedded control systems a component model must focus on extra-functional requirements most often not addressed to the extent required by general purpose component models. This includes requirements on reliability, timing, resource usage and linkage to specific hardware. It might be a trade off between flexible composition which often is in focus by component models and some extra functional properties. For example, it might be a requirement that components for an embedded system must be physically smaller than the common office application component; no unused code can be included since the embedded memory shall be as small as possible.

We are proposing a component model aimed for embedded control systems, addressing the extra functional requirements with focus on real-time analysis. The real-time requirements of a control system can be directly descended from the environment that is controlled. The most common real-time requirement of a control system is to generate a response to an event before a certain point in time, which forms the deadline of the response. To be able to guarantee that the deadline of a certain event can be met during all possible conditions, a schedulabillity analysis is applied. However, analysis of real-time systems can be restrictive since, in order to guarantee the timeliness in the system, it assumes a worst case scenario based on Worst Case Execution Time (WCET) estimations and worst case environment assumptions. Since the worst case scenario in most cases will occur very infrequently, the resource usage at runtime will be lower than estimated.

Consider that some missions in a system can be performed with different quality levels, for instance, monitoring more or less parameters, more or less

deep iterations in numerical approximations, sometimes it may even be possible to not execute a particular task at all and so forth. It would then be possible to adjust the quality level to the available resources and thereby reclaim resources during run-time. The Multiple Versions paradigm [1] is a method that can be used to reduce the consequences of pessimistic analysis of real-time systems. In this paper we are packaging this paradigm into real-time components.

The component model presented is based on ideas from the model described in [2]. Their work defines a prediction enabled component technology (PECT) [3]. In a PECT, both constructive and analytic models are considered. A constructive model deals with functional properties, while an analytic model is concerned with non functional (extra functional) properties, e.g., timing and memory issues.

The contribution of this work is the joining of component technology and theories for obtaining flexible, yet reliable, real-time systems. We have defined a component model that makes this integration possible, while still preserving fundamental component properties. Moreover, we have extended existing work to also allow aperiodic activities, something which is especially important when considering flexible systems.

The rest of the paper is outlined as follows. Section 7.2 presents fundamental RTS theory. In section 7.3, the component model is presented, topics as component description and system assembly are treated. The paper proceeds in section 7.4 by describing the required run-time mechanisms, which is proposed to be built into a middleware. The analysis possibilities are also presented in this section. The last section concludes the paper and contains suggestion to future work.

7.2 Real-Time Systems

In this section, we give an introduction to some basic concepts and principles in real-time systems.

7.2.1 Definition and basic terminology

Real-time systems are computer systems in which the correctness of the system depends not only on the logical correctness of the computations performed, but also on the time factors [4]. A real-time system typically consist of a number of resources (e.g., one or several processors), a number of tasks, each one as-

sociated with a program code, and a scheduler that assigns each task a fraction of the processor(s), according to a scheduling policy. Tasks are usually divided in periodic and non periodic. Periodic tasks consist of an infinite sequence of invocations, called instances or jobs. Non periodic tasks are invoked by the occurrence of an event and are divided into aperiodic and sporadic tasks. While the arrival times of aperiodc tasks are not known in advance, sporadic tasks arrive with a minimum interarrival time, i.e., the minimum time interval between two consecutive invocations. Tasks can have various parameters, such as period, deadline and priority, depending on the scheduling policy chosen to be used. What is common for all real-time tasks is the worst case execution time (WCET), which has to be calculated in order to be able to make any predictions about the system behaviour, i.e., to guarantee that the timing requirements will be met at run-time.

Real-time systems can be classified into two major categories: hard and soft real-time systems. Hard real-time systems are computer systems in which all task deadlines must be met. Examples of such systems are medical control equipment or vehicle control systems. On the other hand, in soft real-time systems, i.e., multimedia applications, a number of deadlines can be missed without serious consequences. In this paper we will primarily focus on hard real-time systems.

The choice of scheduling technique used in order to achieve different requirements has been well analyzed and discussed. Off-line table-driven scheduling is usually used to achieve predictability in systems in which failure may have catastrophic consequences, but for the cost of flexibility as task executions are fixed and determined in advance, and limited ability to handle tasks with incompletely known attributes, e.g., aperiodic or sporadic run-time events. If the main goal is to achieve run-time flexibility, the approach typically used is priority driven scheduling, but the price to pay is the limited ability to handle multiple complex constraints.

7.2.2 Off-line scheduling

Off-line, table driven, scheduling for time-triggered systems provides determinism, as all times for task executions are determined and known in advance. In addition, complex constraints can be solved off-line, such as distribution, end-to-end deadlines, precedence, jitter, or instance separation. The guarantee that tasks will meet their deadlines is the off-line constructed schedule. However, since all actions have to be planned before start-up, run-time flexibility is lacking.

7.2.3 Priority driven (on-line) scheduling

Priority driven (on-line) scheduling can be divided in two main categories: fixed priority scheduling (FPS) or dynamic priority scheduling such as earliest deadline first (EDF). Common for both categories is that the scheduling decision for individual tasks is made at run-time, based on the priority of the tasks. This results in a flexible system with a potentially higher ability to cope with run-time events. Temporal analysis of priority based systems focuses on providing guarantees that all instances of tasks will finish before their deadlines. The actual start and completion times of execution of tasks, are generally not known and depend largely on run-time events.

In fixed priority based systems, guarantees for temporal behaviour are achieved by performing response time analysis (in standard FPS) [5], [6], [7], [8] and [4]. In dynamic priority based systems, i.e., EDF, the guarantee that all tasks will meet their deadlines is the processor utilization, e.g., max 100% [9].

In this paper we will focus on hard, fixed priority driven real-time systems. That is mainly due to the run-time flexibility provided by this kind of systems, wide usage in the industry and low run-time overhead compared to, e.g., EDF-based systems.

7.3 Component Model

In this paper, we deal with the extra-functional requirements of a typical embedded real-time system and add flexibility by introducing services with different quality levels.

The basic idea of the component model is that the services capsulated within a component shall be related in some way, like the methods offered by an ordinary C++ or Java object should be related. A summary of the characteristics of a component is presented below:

- The implementation of a component is not reachable by a third party; a component is a black-box. The only way to communicate with a component is through its interfaces.
- Advanced components can be composed from basic components. It is considered as an advantage in system design to naturally be able to view a larger composition of design.
- A service provided by a component can be implemented in different versions, in direction with the multiple version paradigm, the different ver-

sions are denoted quality levels.

- A service provided by a component can be active, or passive. An active service is scheduled by the run-time system, while a passive service is not directly in contact with the run-time mechanisms.

7.3.1 Component description

In figure 7.1, we present an UML meta-model of a component. The stereotypes show the property class of the building bricks and have the same meaning as in [3], i.e., analytic properties are those needed for analysis and constructive are those that provide functionality.

A component provides one or more services, which is similar to methods offered by an object in an object oriented language as Java or C++.

A service provides in and out ports [10] for exchanging data with other services. Connecting in and out ports is the only way to exchange data between services, even for services within the same component. The number of in and out ports and the type of parameters passed through the ports are free for a component (or service) developer to specify. As mentioned a service can be active or passive, an active service has an associated descriptor containing all parameters needed by the run-time system. Further more, a service has one or more quality levels, which are completely independent procedures for solving the same problem. The number of provided quality levels is free for the developer to specify, but at least one should be provided.

Each quality level has an implementation. A function pointer represents the implementation of the quality level for a basic component. However, a quality level provided by a more advanced composed component has a sequence of sub-services, which should be executed upon an invocation. A quality level also has a WCET and a value associated with itself. Upon a concretization of the model, it is possible to add more parameters, such as static memory consumption, depending on desired tools and focuses. The WCET represents the maximum time interval for which the service is executed as a sequential program without being interrupted. Theories regarding WCET estimation have been presented ([11] [12]). The value can be set to an arbitrary number, and is used by the run-time system for choosing between different quality levels. Short and a bit simplified, the run-time system tries to collect as high value as possible depending on available resources. If WCET and available time allows the version with highest value will be chosen for execution.

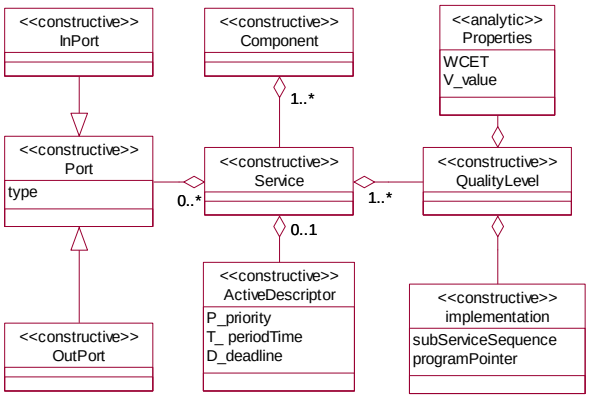

Figure 7.1: UML meta-model of a component

7.3.2 Component Interfaces

The interface of a component is defined as a specification of its access points, as in [13]. A component has multiple access points specified as detached independent interfaces. The interfaces are the only visible parts of a component; they offer no implementation of the services, but only methods and protocols to access the services. Furthermore it is assumed that each interface, besides the pure functional specification, provides all necessary information regarding the provided service in the form of ordinary comments added by the developer. Included additional information should at least be semantics describing the service, and, especially for RTS, temporal and memory requirement attributes. The proposed component model consists of three types of interfaces.

- Data interfaces, are port based, and contain information about existing ports and data type definitions. Each service can provide in and out ports for sharing data between each others.

- Control interfaces, provide access points for control of a component. A control interface provides methods for invocation of the different encapsulated services and also if a service is defined as active it consists of parameters and structures required by the run-time system.

- Analytic interfaces, provide parameters concerning different quality levels of a service. The minimum amount of parameters included is the number of levels with corresponding WCET and value. This interface substitute basis for the decisions made by the run-time system.

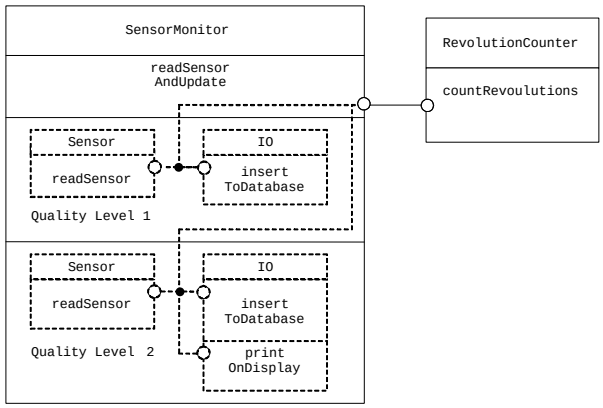

Figure 7.2: data connections through in and out ports

7.3.3 Assembling components

Assembling components or parts of components into new components and assembling systems through exchanging data between components is carried out utilizing different interfaces. Hence, data exchange and component composition are independent to each others. A consequence is that data can cross component boundaries utilizing the same mechanisms as within a component.

Exchanging data between services is carried out by connecting in ports and out ports. A smallest requirement on the type level is that the interacting in and out ports uses the same types of parameters. The ports of a particular service is accessible trough the data interface provided by the component hosting the service. When a service is launched, it begins with reading data from all its in ports (zero or more), and, when its execution is finished, data is written to its out ports (zero or more). A schematic picture of the dataflow between the components SensorMonitor and RevolutionCounter is shown in figure 7.2. In the

figure the interior of the composed component SensorMonitor is hatched, the in ports in the figure are marked as small circles to the left of each component, and the out ports are marked as a small circle to the right of each component. The figure describes the connections between in and out ports for the implementation of two different quality levels of the readSensorAndUpdate service, provided by the SensorMonitor component. As shown the connections of in and out ports are transparent across the node boundaries.

Composing a service within a component from services provided by other components is achieved through the definition of a sequence. The sequence is constructed by using the control interface provided by the subcomponents. A pseudo code example is shown in figure 7.3; it could be the sequence belonging to the SensorMonitor component shown in figure 7.2. Quality level 1 of the provided readSensorAndUpdate service is built by the two sub services readSensor and insertToDatabase provided respectively by the sensor component and the IO component. Quality level 2, however, utilizes an additional service printOnDisplay. As shown in the pseudo code example (figure 7.3), choosing quality level of sub services when defining a sequence utilizes static binding, e.g., the desired quality level is specified. To be able to utilize different quality levels of a sub service, a designer must additionally compose different top quality levels.

```
SensorMonitor Sequences{
                readSensorAndUpdate{
                QLevel1{
                                Sensor.Control.readSensor.QLevel1();
                                IO.Control.insertToDatabase.QLevel1();
                }
                QLevel2{
                                Sensor.Control.readSensor.QLevel1();
                                IO.Control.insertToDatabase.QLevel1();
                                IO.Control.printOnDisplay.QLevel1();
                }
}
```

Figure 7.3: data connections through in and out ports

Assembly of applications may be viewed as a hierarchy. Basically, we can distinguish between:

- Basic passive components
- Composed passive components
- Active components

- An application

Basic passive components are components which are rather small and offer well defined services. On this level WCET estimations of each service should be included. It is usually easier to make an accurate WCET estimation on a small service than on complex composed services. Once the estimated value is assigned, it can be reused in other systems. However, the WCET can be further tuned during the life of a basic component through an evolutionary process.

Composed passive components are components which consist of services derived by specifying sequences of other services, and internal data connections through connecting the in and out ports of the included services. WCET is automatically calculated from the WCET:s of the included services. The automatic calculation is based on the serialization pattern naturally provided by the sequence; it is basically a summation of the WCET:s of the included services, since no concurrent services are regarded. Instead, the dynamics with concurrent services are taken care of by the schedulability analysis described in the next section.

Active components have services that are scheduled by the underlying component technology. Active services have some additional parameters asserted by the developer when the service is chosen to be active. The parameters are (T, D, P). T is the period time, and is asserted for periodic invocation of a service. If T is set, the service will be automatically set as ready for execution with an interval T between two invocations. The relative deadline upon an invocation of a service is denoted by D, and represents the time interval from which the service became ready for execution until it has to be completed. P is priority and is a number representing the priority of the service; it can be asserted with an arbitrary theory.

Finally, the application is the set of active services, which together solves the particular mission for the system. A set of active services can be compared with a set of tasks in a traditional RTS.

7.4 Component technology

The component model offers a set of quality levels. Temporal analysis of fixed priority pre-emptive systems [6] can guarantee temporal behaviour before run-time. In the model proposed in section 7.3, a service has a number of quality levels. One of the levels is the basic level that has been guaranteed pre-runtime by a schedulability test. However the schedulability test is based on the WCET that can be over estimated. Hence, the pre-run-time analysis can turn quite

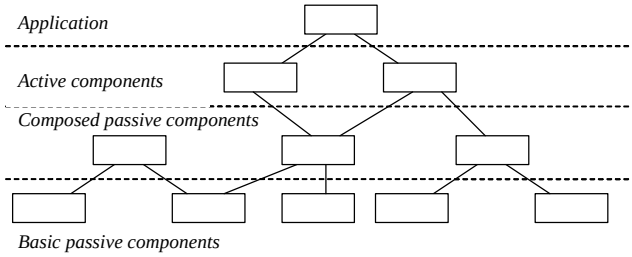

Figure 7.4: hierarchy of components building an application

pessimistic. Because of this pessimistic off-line analysis the component model also offers the same service at a higher quality level. The service can be performed with higher quality if there are more available resources than the basic quality level demands. The concept of quality is up to the designer to define. However, a higher level of quality would probably demand more resources in form of processor time. In [14], the authors have compared several admission policies to be used with Multiple Versions paradigm [1]. In this paper we have adapted and extended the Adaptive Threshold algorithm to suit our component model.

7.4.1 Runtime system

The run-time system is a middleware, intended to give the user an image of a totally self-providing service i.e., a service that automatically chooses to run the highest quality level with respect to available resources. The service scheduling algorithm is located in the middleware, hence it is easier to have a dispatcher suited for a specific system or underlying scheduling algorithm without having to change the service or component model.

7.4.2 Pre-runtime analysis

Real-time analysis is an important tool to examine if a set of tasks is feasible, without having to try every possible execution path. The analysis is used to ensure that all deadlines and other extra functional properties are met.

In our approach, the schedulability analysis is performed in order to guarantee the basic level for each periodic service. There are many ways of for-

mally guaranteeing that a set of services will complete before their deadlines. In [5] a formal analysis for guaranteeing services in fixed priority systems is proposed. If the schedulability analysis fails, the system has to be redesigned by, e.g., choosing a basic quality level with a lower WCET on one or several services.

As previously mentioned, the off-line analysis is quite pessimistic and during run-time the services will usually not use all time allotted.

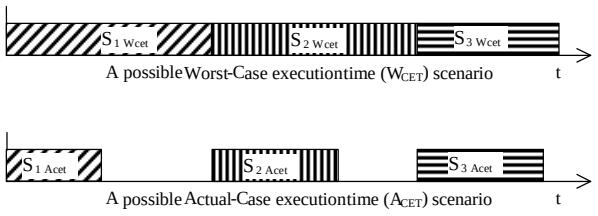

Figure 7.5: Worst case vs. actual case execution times

A method for using the time gap between the WCET and the actual case execution time (ACET) is known as resource reclaiming. The ACET is, as the name reveals, the amount of time actually used at run- time for executing a service.

Assume a set of components that have been analysed pre-run time with respect to their basic quality levels. If the WCET:s have been over estimated, the ACET:s will be lower at run-time. The difference in time between the estimated WCET and ACET of a service is here called spare time.

In figure 7.6, the second service has executed less than WCET. The subsequent component then chooses to run at a higher quality level (see figure 7.7) since the time available is greater than in the original schedule.

However, in figure 7.7, the WCET of the higher quality level of the third service is also an approximation. Consequently the ACET of that service is likely to be lower than it's WCET, which is the case in figure 7.7. This result is spare time, just as with the second service (figure 7.6). The spare time of the third service can be applied on the next service and so forth.

Another way to get more time for a service is to postpone lower prioritized services. Postponing lower quality services can be acquired through, e.g., the slack-stealing algorithm [15]. The admission algorithm Adaptive Threshold considers, e.g., resource reclaiming and slack stealing. For off-line scheduling

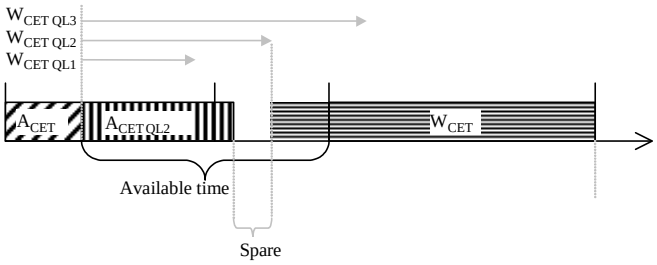

Figure 7.6: ACET allows a subsequent component to run at a higher quality level

based systems, a method to address the issue of resource reclaiming has been presented in [16].

The "left over" time, which we here choose to denote slack, that consists of reclaimed resources and slack by postponing lower prioritised services, can be used for executing components at a higher quality level. In order to schedule a component at a higher quality level, the run-time system must first decide if there's enough slack. The basic idea is to query each quality-level for its WCET, and compare the amount of available time with the time required for a specific quality-level.

Small resource limited systems often consist of only a while loop. Hence, a relatively large complex algorithm might not be feasible. However, the component model can be used for an arbitrary complex system. In this paper we focus on an algorithm aimed for complex embedded systems.

7.4.3 On line service scheduler

The on line scheduler is based on the Adaptive Threshold approach [14]. We first apply the adaptive threshold algorithm to our component model. Then, in section 7.4.3 we extend it to handle aperiodic services. Last, in section 7.4.3, we illustrate our algorithm by an example.

Adaptive threshold

The service scheduler assumes that each service has at least one basic quality level. The pre-run-time schedule is analysed with respect to the basic quality

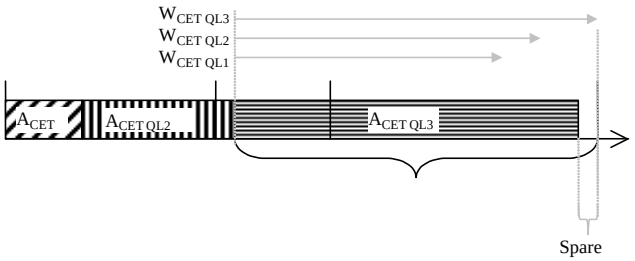

Figure 7.7: A second subsequent components that executes at a higher quality level

level of each scheduled service. We refer to these quality levels as QL_s^i where i and s are the level and service numbers respectively. Hence, quality levels two of service one would be denoted as QL_2^1. Each quality level has a worst-case execution time $WCET_s^i$ and a value V_s^i which is set pre runtime. The service incorporates a deadline D_s, and a release time R_s. The basic quality level of a service is denoted as QL_s^P and is as mentioned earlier guaranteed via off-line analysis.

In [14], a method is presented to calculate the minimum processing time, $A_s(t)$, available before the deadline of a service (equation 7.1).

$$A_s(t) = Wcet_s^P + min[L_i(t, D_s), min_{\forall j \in lp(i)} S_j(t)] \tag{7.1}$$

In equation 7.1, L_i(t,D_i) is the lower bound on the additional execution time available at priority level i in the interval [t , t + D_i). S_j is the extra interference that any task with priority less than i can be subjected to without missing its next deadline.

As equation 7.1 shows, at time t, the time interval available for executing a medium priority service (MP) is L_i. That is the additional execution time, i.e., the time not used by any service. The reclaimed time can be used for any service ready to execute at time t. S_j, is the time that can be allocated through delaying lower prioritized services without missing their deadlines. However the maximum amount of time that can be used by MP is until its deadline (d_{MP}).

In addition to the values Vsi corresponding to each service, there is also a global system value V^{SYS}. The V^{SYS} is the mean value of executed services.

In [14], a value density-based strategy has been proposed. The strategy is used to choose the quality level that gives the highest value density. The value-density D_s^i is given by:

$$D_s^i = \frac{V_s^i \cdot Wcet_s^i + [A_s(t) - Wcet_s^i] \cdot V^{SYS}}{A_s(t)} \tag{7.2}$$

The quality level with the highest value density D_s^i is chosen for execution. For all components that have been analysed off-line, there is at least one feasible quality level, i.e., the basic quality level.

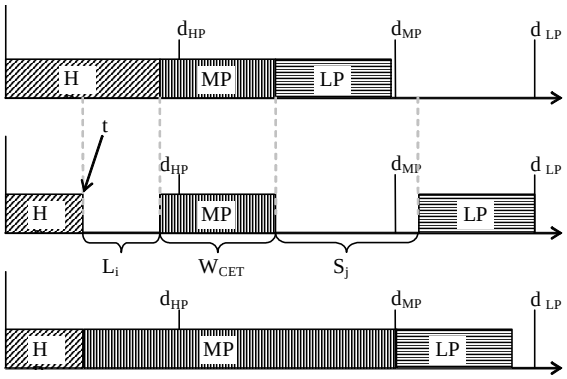

Figure 7.8: visual explanation of equation 7.1

Aperiodic requests extension

In this section we will describe an extension to the adaptive threshold algorithm. In particular, we want to handle run-time events with incompletely known parameters, e.g., aperiodics.

Aperiodic requests have to be considered in a number of priority based real-time systems, as not all event parameters can be known beforehand. In some cases it might be of greater value to the system to execute an aperiodic service, rather than executing a periodic service at a higher quality level. However, all the periodics have to be guaranteed to complete before their deadlines. The value of an aperiodic service is decided in the same way as the value of the periodic services, i.e., based on the value density. Hence, the algorithm for the

value density has to be modified to include aperiodics as well. A value V has to be added to aperiodic services such that the system accrues a higher value. The computation of the value density for all pending periodic and aperiodic requests is described in equation 7.3.

$$D_A^i = \frac{V_A^i \cdot Wcet_A^i + V_s^i \cdot Wcet_s^i + \lfloor A_s(t) - Wcet_s^i - Wcet_A^i \rfloor \cdot V^{SYS}}{A_s(t)} \tag{7.3}$$

However, the periodic services must always be guaranteed. We assume that the aperiodic services have the same structure as the periodics, i.e., they have at least a basic quality level. Hence, an acceptance test for the aperiodic services must be included. The WCET of an aperiodic service's basic quality level is compared to the available time. However, since the periodic services must be guaranteed to run, the available-time for the aperiodic services is not quite the same as for the periodic. Considering equation 7.3, the WCET corresponding to the basic quality level is the time that the periodic service has been guaranteed through the off-line analysis. That time cannot be allocated by aperiodic services, thus:

$$A_p(t) = min[L_i(t, D_s), min_{\forall j \in lp(i)} S_j(t) \tag{7.4}$$

In equation 7.4, $A_p(t)$ is the time that can be allocated to any aperiodic quality level with a WCET lower than or equal to $A_p(t)$.

Example

In this section we present an example of the adaptive threshold algorithm with the aperiodic request extension.

We assume a service S with three quality levels QL_1, QL_2 and QL_3. We assume the following worst-case execution times, $Wcet_s^1$=1, $Wcet_s^2$=4, $Wcet_s^3$ = 9 and the values V_s^1=1, V_s^2=8, V_s^3=15. We also assume $A_s(t)$=10 and quality level 2 is the basic level i.e. the level that has been analysed and guaranteed pre-runtime. All quality levels are feasible, thus the algorithm will choose the quality level that will accrue the greatest value. We will now look at a few scenarios where the algorithm will choose different quality levels.

Scenario 1: We assume V^{SYS}=5 and no aperiodic services. Executing quality level 1 for 1 time unit will accumulate a value of one. The rest of the time (nine time units) will give a value of 45 because the V^{SYS} (9*5=45), hence the total value when choosing quality level one will be 1+45= 46. A

value of 46 will give a value density of 4.6. In the same way QL_2 will give a value of 8+30=38, thus a value density 3.8. QL_3 will acquire a value of 15+5=20 and a value density of 2. Consequently with V^{SYS} equal to five, quality level one is the level that will accrue the highest value density.

Scenario 2: Here we assume V^{SYS}=2 and no aperiodic services. In the same way as scenario 1, quality level one will consume one time unit and accumulate 1 to the value. The remaining 9 time units will be used for the other services in the system which will accumulate a value of 18, hence a total accumulated value of 19. QL_2 will accumulate a value of eight for the first 4 time units, and a value of 12 for the following 6 time units, hence a total of 20. QL_3 will acquire a value of 17. Consequently QL_2 is the best choice for this scenario.

Scenario 3: We assume V^{SYS}=1 and the aperiodic service S_A with two quality levels S_A^1 and S_A^2. The properties of $SA1$ are: $WCET_A^1$=1 and V_A^1=3. The properties of S_A^2 are $WCET_A^2$=3 and V_A^2=6. Further we assume that the available-time for aperiodic requests A_p=6, because WCET of the basic quality level is guaranteed offline and cannot be allocated by an aperiodic service. One can see that the possible permutations for executing the periodic and aperiodic service are

- V_S^1+V_A^1+8*V^{SYS} = 1+3+8=12 (1.2)
- V_S^1+V_A^2+6*V^{SYS} = 1+6+6=13 (1.3)
- V_S^2+V_A^1+5*V^{SYS} = 8+3+5=16 (1.6)
- V_S^2+V_A^2+3*V^{SYS} = 8+6+3=17 (1.7)
- V_S^3+V_A^1 = 15+3=18 (1.8)

One can see directly that QL_3 of the periodic service and QL_1 of the aperiodic service will acquire the highest value density.

7.5 Conclusions and future work

In this paper we presented work to show that a real-time component is a suitable package for the multiple versions paradigm. We have proposed a component model with a middleware aimed for execution on top of an RTOS, which gives the developer possibilities for issuing real-time guarantees with additional flexibility through implementing multiple versions.

Our component model can be used together with different existing real-time scheduling methods to achieve flexibility, still guaranteeing timeliness. In this paper we show how the model can be used with the Adaptive Threshold algorithm. Furthermore, the Adaptive Threshold algorithm is extended to also cater for aperiodic activities, which is important in order to provide flexibility in many real-time systems.

A prototype implementation of the proposal with development tools and possibility to compile for execution upon some commercial real-time operating system is be the next step towards a realization of the model. Trying to utilize such a prototype in the development of an embedded control system, would result in useful input for future development of the component model.

Bibliography

[1] J. A. Stankovic and K. Ramamritham. What is predictability for real-time systems? *Real-Time Systems*, 2(4), November.

[2] A. Wall, M. Larsson, and C. Norstrom. Towards an impact analysis for component based real-time product line architectures. In *Proceedings of the 28th Euromicro Conference*, pages 81–88, 2002.

[3] S. A. Hissam, G. A. Moreno, J. Stafford, and K. C. Wallnau. Packaging predictable assembly with prediction-enabled component technology. Technical report, November 2001.

[4] L. Sha, R. Rajkumar, and J. Lehoczky. Task period selection and schedulability in real-time systems. *IEEE Transactions on Computer*, 39(9), September 1999.

[5] N. C. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical report, Technical Report, Department of Computer Science, University of York, 1991.

[6] N. C. Audsley, A. Burns, M. F. Richardson, K. Tindell, and A. J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284–292, september 1993.

[7] J. C. Palencia and M. Gonzalez Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *Proc. of the 19th Real-Time Systems Symposium*, pages 26–37, December 1998.

[8] K. Tindell. Adding time offsets to schedulability analysis. Technical report, Technical Report, Department of Computer Science, University of York, 1994.

[9] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in hard-real-time environment. *Journal of the Association for Computing Machinery*, 20(1):46–61, 1973.

[10] D. B. Stewart, R. A. Volpe, and P. K. Khosla. Design of dynamically reconfigurable real-time software using port-based objects. In *IEEE Transactions on Software Engineering*, pages 759–776. IEEE, December 1997.

[11] J. Engblom, A. Ermedahl, M. Sjödin, J. Gustafsson, and H. Hansson. Execution-time analysis for embedded real-time systems. In *proceedings of Software Tools for Technology Transfer*, pages 1080–1812, 2000.

[12] P. Puschner. A tool for high-level language analysis of worst-case execution times. In *Proceedings of 10th Euromicro Workshop on Real-Time Systems*, pages 130 –137, 1998.

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

[14] N. Audsley R. Davis, S. Punnekkat and A. Burns. Flexible scheduling for adaptable real-time systems. *Proceedings of IEEE Real-Time Technology and Applications Symposium*, May 1995.

[15] J. P. Lehoczky and S. Ramos-Thuel. An optimal algorith for scheduling soft-aperiodic tasks fixed priority preemptive systems. In *Proceedings Real-Time System Symposium*, pages 110–123. IEEE, Decembed 1992.

[16] T. Lenvall, G. Fohler, and B. Lindberg. Handling aperiodic tasks in diverse real-time systems via plug-ins. In *Proceedings of the 5th IEEE International symposium of Object-Oriented Real-Time Distributed Computing*, 2002.

Chapter 8

Paper C: Introducing a Component Technology for Safety Critical Embedded Real-Time Systems

Kristian Sandström, Johan Fredriksson and Mikael Åkerholm
In International Symposium on Component-Based Software Engineering (CBSE7)
Edinburgh, Scotland, May 2004

Abstract

Safety critical embedded real-time systems represent a class of systems that has attracted relatively little attention in research addressing component based software engineering. Hence, the most widely spread component technologies are not used for resource constrained safety critical real-time systems. They are simply to resource demanding, to complex and to unpredictable. In this paper we show how to use component based software engineering for low footprint systems with very high demands on safe and reliable behaviour. The key concept is to provide expressive design time models and yet resource effective run-time models by statically resolve resource usage and timing by powerful compile time techniques. This results in a component technology for resource effective and temporally verified mapping of a component model to a commercial real-time operating system.

8.1 Introduction

The vehicle domain represents a class of embedded real-time systems where the requirements on safety, reliability, resource usage, and cost leaven all through development. Historically, the development of such systems has been done using only low level programming languages, to guarantee full control over the system behaviour. As the complexity and the amount of functionality implemented by software increase, so does the cost for software development. Therefore it is important to introduce software development paradigms that increase software development productivity. Furthermore, since product lines are common within the domain, issues of commonality and reuse is central for reducing cost as well as increasing reliability.

Component based software engineering is a promising approach for efficient software development, enabling well defined software architectures as well as reuse. Although component technologies have been developed addressing different demands and domains, there are few component technologies targeting the specific demands of safety critical embedded real-time systems. Critical for the safe and reliable operation of these systems is the real-time behaviour, where the timeliness of computer activities is essential. To be able to guarantee these properties it is necessary to apply real-time systems theory. Thus, a component technology to be used within this domain has to address specification, analysis, and implementation of real-time behaviour.

A typical real-time constraint is a deadline on a transaction of co-operating activities. A transaction in these systems would typically sample information about the environment, perform calculations based on that information and accordingly apply a response to the environment, all within a limited time frame. Also important is the ability to constrain the variation in periodicity of an activity (jitter). The reason for this is that variations in periodicity of observations of the environment and responses to the same, will affect the control performance. Hence, a component technology for this domain should have the ability to clearly express and efficiently realize these constraints [1],[2],[3],[4].

The work described in this paper present a component technology for safety critical embedded real-time systems that is based on experience from our previous work with introducing state-of-the-art real-time technology in the vehicle industry. The benefits in development have been discussed in [5] and have also been proven by long industrial use. That real-time technology has been incorporated in the Rubus development suite and has been further developed [6]. Experience from the industrial application of the research reveals that a proper component model is not enough; success requires an unbroken chain of

models, methods, and tools from early design to implementation and run-time environment.

The contribution of the work presented in this paper includes a component technology for resource effective and temporally verified mapping of a component model to a resource structure such as a commercial Real-Time Operating System (RTOS). This is made possible by introduction of a component model that support specification of high level real-time constraints, by presenting a mapping to a real-time model permitting use of standard real-time theory. Moreover, it supports synthesis of run-time mechanisms for predictable execution according to the temporal specification in the component model. Furthermore, in this work some limitations in previous work with respect to specification and synthesis of real-time behaviour are removed. These limitations are partially discussed in [5] and is mainly related to jitter and execution behaviour.

Many common component technologies are not used for resource constrained systems, nor safety critical, neither real-time systems. They are simply to resource demanding, to complex and unpredictable. The research community has paid attention to the problem, and recent research has resulted in development of more suitable technologies for these classes of systems. Philips use Koala [7], designed for resource constrained systems, but without support for real-time verification. Pecos [8] is a collaboration project between ABB and University partners with focus on a component technology for field devices. The project considers different aspects related to real-time and resource constrained systems, during composition they are using components without code introspection possibilities that might be a problem for safety critical applications. Rubus OS [6] is shipped with a component technology with support for prediction of real-time behaviour, though not directly on transactions and jitter constraints and not on sporadic activities. Stewart, Volpe, and Khosla suggest a combination of object oriented design and port automaton theory called Port Based Objects [9]. The port automaton theory gives prediction possibilities for control applications, although not for transactions and jitter constraints discussed in this paper. Schmidt and Reussner propose to use transition functions to model and predict reliability in [10]; they are not addressing real-time behaviour. Wallnau et al. suggest to restrict the usage of component technologies, to enable prediction of desired run-time attributes in [11], the work is general and not focused on particular theories and methods like the work presented in this paper.

The outline of the rest of this paper is as follows; section 8.2 gives an overview of the component technology. In section 8.3 the component model is described and its transformation to a real-time model is explained in section

8.4. Section 8.5 presents the steps for synthesis of real-time attributes and discusses run-time support. Finally, in section 8.6, future work is discussed and the paper is concluded.

8.2 Component Technology

In this section we will give an overview of the component technology facilitating component based software development for safety-critical embedded real-time systems. We will hereafter refer to this component technology as the AutoComp technology. A key concept in AutoComp is that it allows engineers to practise Component Based Software Engineering (CBSE) without involving heavy run-time mechanisms; it relies on powerful design and compile-time mechanisms and simple and predictable run-time mechanisms. AutoComp is separated into three different parts; component model, real-time model and run-time system model. The component model is used during design time for describing an application. The model is then transformed into a real-time model providing theories for synthesis of the high level temporal constraints into attributes of the run-time system model. An overview of the technology can be seen in Figure 8.1. The different steps in the figure is divided into design time, compile time, and run-time to display at which point in time during development they are addressed or used.

During design time, developers are only concerned with the component model and can practise CBSE fully utilizing its advantages. Moreover, high level temporal constraints in form of end-to-end deadlines and jitter are supported. Meaning that developers are not burdened with the task of setting artificial requirements on task level, which is essential [12], [5]. It is often natural to express timing constraints in the application requirements as end-to-end constraints.

The compile time steps, illustrated in Figure 8.1, incorporate a transition from the component based design, to a real-time model enabling existing real-time analysis and mapping to a RTOS. During this step the components are replaced by real-time tasks. Main concerns in this phase are allocation of components to tasks, assignment of task attributes, and real-time analysis. During attribute assignment, run-time attributes that are used by the underlying operating system are assigned to the tasks. The attributes are determined so that the high level constraints specified by the developer during the design step are met. Finally, when meeting the constraints of the system, a synthesis step is executed. It is within this step the binary representation of the system is cre-

ated, often the operating system and run-time system are also included with the application code in a single bundle

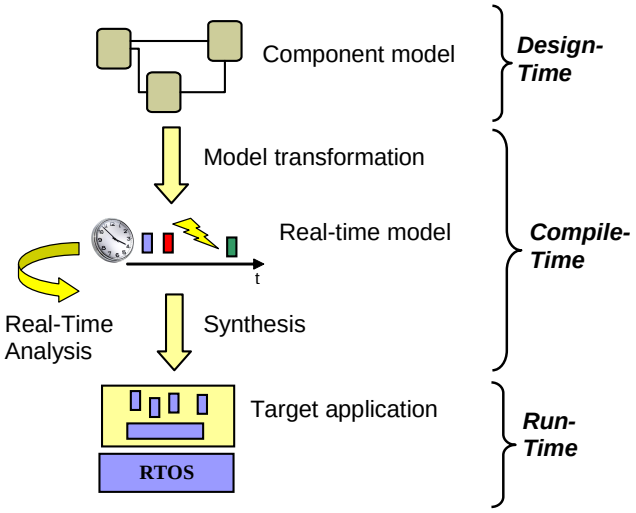

Figure 8.1: The AutoComp component technology

The run-time system is assumed to be a traditional RTOS with Fixed Priority Scheduling (FPS) of tasks. Most commercial RTOS can be classified into this category; furthermore they are simple, resource efficient and many real-time analysis techniques exist. In some cases a layer providing run-time support for the tasks has to be implemented in order to fully support FPS models used in real-time theory.

8.3 Component Model

Vehicles present a heterogeneous environment where the interaction between the computer system and the vehicle take different forms. Some vehicle functionality requires periodic execution of software, e.g., feedback control, whereas other functionality has a sporadic nature, e.g., alarms. Although vehicle control plays a central role, there is also an abundance of other functionality in vehicles that is less critical and has other characteristics, e.g., requires more flexibility. Although less critical, many of these functions will still interact

with other more critical parts of the control system, consider for example diagnostics. We present a model that in a seamless way allows the integration of different functionality, by supporting early specification of the high level temporal constraints that a given functionality has to meet. Moreover, the computational model is based on a data flow style that results in simple application descriptions and system implementations that are relatively straightforward to analyse and verify. The data flow style is commonly used within the embedded systems domain, e.g., in IEC 61131 used for automation [13] and in Simulink used for control modelling [14].

The definition of the AutoComp component model is divided into components, component interfaces, composition, the components invocation cycle, transactions and system representation. In Figure 8.2 the component model is illustrated using UML2, which could be a possible graphical representation during design.

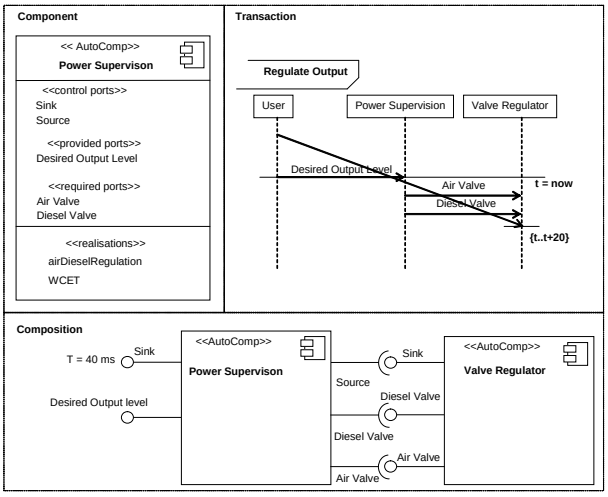

Figure 8.2: In the upper left part of the figure there is a UML 2 component diagram for modelling of a component. The lower part of the figure is a composition diagram showing a composition of two components. Finally the upper right part of the figure is a sequence diagram with a timing constraint that is used to express the end-to-end deadline for a transaction

The components are defined as *glass box*, meaning that a developer can see the code of a component for introspection purposes. It does not mean that a developer has to look into a component during normal composition, and not that it is allowed to modify a component. The introspection possibility is a requirement during verification of safety critical applications in order to gain complete knowledge about components behaviour. Furthermore, the components can only exchange data with each others through data ports. A component can be a composite containing a complete subsystem, or a basic component with an entry function. Composite components can be treated as any other component during composition, but it is also possible to enter a composite and change timing requirements and other properties. The entry function provided by non-composite components can be compared to the entry function for a computer program, meaning that the contained number of functions of the component can be arbitrary.

The interfaces offered by a component can be grouped into the two classes data and control interfaces. The data interfaces are used to specify the data flow between components, and consist of data ports. Data ports have a specified type and can be either provided or required. Provided ports are the ports provided by components for input, i.e., the ports a component reads data from. Required ports are the ports a component writes data to. A component also has a control interface with a mandatory control sink, and an optional control source. The control interface is used for specifying the control flow in the application, i.e., when or as a response to what component should be triggered. The control sink is used for triggering the functionality inside the component, while the control source is used for triggering other components.

During composition the developer has three main techniques to work with. The data flow is specified through connection of provided and required data ports. The rules are as follows; required ports must be wired to provided ports with a compatible type. It is possible to make abstractions through definition of composite components. Composite components can be powerful abstractions for visualizing and understanding a complex system, as well as they provide larger units of reuse. The control flow is specified through binding the control sinks to period times for periodic invocation, to external events for event invocation, or to control sources of other components for invocation upon completion of the other components.

A components invocation cycle can be explained as in the following sentences. Upon stimuli on the control sink, in form of an event from a timer, an external source or another component; the component is invoked. The execution begins with reading the provided ports. Then the component executes the

contained code. During the execution, the component can use data from the provided ports and write to the required ports as desired, but the writes will only have local effect. In the last phase written data become visible on the required ports, and if the control source in the control interface is present and wired to the control sink of another component stimulus is generated.

Transactions allow developers to define and set end-to-end timing constraints on activities involving several components. A transaction in AutoComp can be defined as:

A transaction Tr_i is defined by a tuple $< C, D, J_s, J_c >$ where:

C - represent an ordered sequence of components;

D - represent the end-to-end deadline of the transaction;

J_s - represent the constraint on start jitter of the transaction;

J_c - represent the constraint on completion jitter of the transaction.

The end-to-end deadline is the latest point in time when the transaction must be completed, relative to its activation. Jitter requirements are optional and can be specified for transactions involving time triggered components. Start jitter is a constraint of the periodicity of the transactions starting point, while completion jitter is a constraint on the periodicity of a transactions completion point. Both types of jitter are expressed as a maximum allowed deviation from the nominal period time. A restriction, necessary for real-time analysis, is that components directly triggered by an external event can only be part of a transaction as the first component.

A system can be described with the UML class diagram in Figure 8.3. A system is composed of one or several components, each with a data interface, a control interface and a realization as a subsystem or an entry function. A system also has zero or more data couplings, describing a connected pair of required and provided data ports. Furthermore, systems have zero or more control couplings which describe a connected pair of control sink and source. Finally, the last part of a system is zero or more transactions with the included components, an end-to-end deadline and the possibility to specify jitter requirements.

8.4 Model Transformation

Model transformation involves the steps necessary in order to transit from the component model allowing an efficient and powerful design phase, to a run-

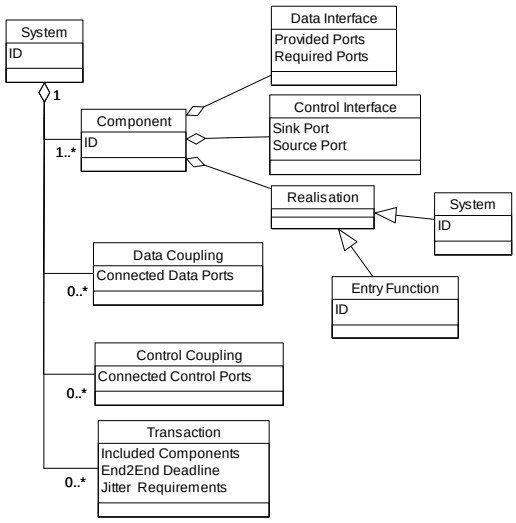

Figure 8.3: UML class diagram showing the static view of the component model

time model enabling verification of temporal constraints and usage of efficient and deterministic execution environments. As previous stated in section 8.2 we assume a FPS run-time model. The FPS model defines a system as a set of tasks with the attributes period time, priority, offset, and WCET. Hence, it is necessary to translate the component model with its temporal constraints in to tasks holding these attributes. The translation is performed in two separate steps; the first step is to make a transformation between components and task (task allocation), the second step is to assign attributes to the tasks (attribute assignment). To assign the FPS model attributes in such a way that the high level temporal constraints on transactions are met is non-trivial and has been addressed in research by e.g., [1], [3].

8.4.1 Task Allocation

The easiest approach for task allocation is a one to one relationship between components and tasks, but that is not necessarily optimal. In fact the task

allocation step has a lot of different tradeoffs. Such tradeoffs can be found between reliability and run time overhead; few tasks reduce run time overhead at the cost of memory protection (usually at task level) between components. Testability and schedulability are examples of other properties that are affected by the allocation scheme.

In this paper we introduce a task allocation strategy that strives to reduce the number of tasks considering schedulability and reliability. Components are not allocated to the same task if schedulability is obviously negatively affected and structurally unrelated components are not allocated to the same task in order to cater for memory protection and flexibility.

The first step in the allocation process is to convert all composite components to a flat structure of the contained basic components. Secondly the following rules are applied:

1. All instances of components are allocated to separate tasks, Worst Case Execution Time (WCET) is directly inherited from a component to the corresponding task

2. The start jitter Js corresponding to a transaction with jitter requirements is set as a requirement on the task allocated for the first component in the ordered sequence C, while the completion jitter Jc is set to the task allocated for the last component in the sequence

3. Tasks allocated for components with connected pairs of control sink and sources, where the task with the source do not have any jitter requirements, and both tasks are participating in the same and only that transaction are merged. The resulting WCET is an addition from all integrated tasks WCET

4. Tasks allocated for time triggered components that have the same period time, not have any jitter constraints and are in a sequence in the same and only that transaction are merged. The resulting WCET is an addition from all integrated tasks WCET

The situation after application of the allocation rules is a set of real-time tasks. The high level timing requirements are still expressed in transactions, but instead of containing an ordered set of components a transaction now contain an ordered set of tasks. The rest of the attributes, those that cannot be mapped directly from the component model to the real-time model are taken care of in the following attribute assignment step. In Figure 8.4, given the two transactions

$Tr_1 =< C, D, J_s, J_c >=< A, B, C, 60, -, 25 >$ and $Tr_2 =< C, D, J_s, J_c >=< D, E, F, 40, 5, - >$ the task allocation step for the components in Table 8.1 is shown. The resulting task set is in Table 8.2.

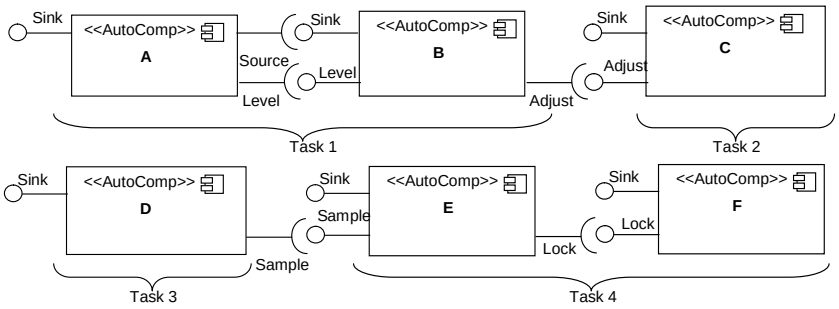

Figure 8.4: Task allocation example

	Sink Bound To	WCET
A	T = 100	5
B	A.Source	10
C	T = 60	5
D	T = 40	5
E	T = 40	6
F	T = 40	9

Table 8.1: A component set

8.4.2 Attribute Assignment

After the components have been assigned to tasks, the tasks must be assigned attributes so that the high level temporal requirements on transactions are met. Attributes that are assigned during task allocation are WCET for all tasks, a period time for periodic tasks and a Minimum Interarrival Time (MINT) for event triggered tasks.

The scheduling model that is used throughout this paper is FPS, where tasks have their priorities and offsets assigned using an arbitrary task attribute

	Trigger	Jitter	WCET
Task 1	T = 100		15
Task 2	T = 60	25	5
Task 3	T = 40	5	5
Task 4	T = 40		15

Table 8.2: The resulting task set

assignment methodology. Examples of existing methods that can be used for priority assignment are Bate and Burns [1], Sandström and Norström [3] or by combination of Yerraballi [15] or Cheng and Agrawala [16] with Dobrin, Fohler and Puschner [17]. In this paper it is assumed that task attributes are assigned using the algorithm proposed by Bate and Burns [1], and it is showed that the component model described in this paper is applicable to their analysis model. Weather the tasks are time triggered or event triggered is not considered in the Bate and Burns analysis but is required during the mapping to the FPS model, where periodic and event triggered (sporadic) tasks are separated. The attributes that are relevant, considering this work, in the Bate and Burns approach are listed below.

For tasks:

T (Period) - All periodic tasks have a period time that is assigned during the task allocation. Sporadic tasks have a MINT that analytically can be seen as a period time;

J (Jitter) - The jitter constraints for a task is the allowed variation of task completion from precise periodicity. This type of jitter constraint is known as completion jitter. Jitter constraints can be set on the first and last task in a transaction;

R (Worst Case Response time) - The initial Worst Case Response time for a task is the WCET for the task, i.e., the longest time for a task to finish execution from its starting point in time.

For transactions:

T (Period) - The period of a transaction is the least common multiple of the period times of the participating tasks of the transaction;

End-to-End deadline - Transactions have a requirement that all tasks have finished their execution within a certain time from the transactions point of start in time.

In Bate and Burns approach additional attributes, such as deadline and separation for tasks and jitter requirements for transactions are considered. In this paper those attributes are disregarded since there are no such requirements in the previously described component model. It is trivial to see that from the component model, the period and jitter constraints match the model proposed by Bate and Burns. The initial worst case response time R is assigned the WCET value in the component model. For the transaction the end-to-end deadline requirements match the transaction deadline of the Bate and Burns model. The period time of the transaction is derived from the least common multiple of the period of the tasks participating in the transaction.

The next step is naturally to assign the FPS model with run-time and analysis attributes. The new attributes priority and offsets will be derived through existing analysis methods [1]. The new parameters for the FPS model are described below.

P (Priority) - The priority is an attribute that indicates the importance of the task relative to other tasks in the system. In a FPS system tasks are scheduled according to their priority, the task with the highest priority is always executed first. All tasks in the system are assigned a priority;

O (Offset) - The offset is an attribute that periodic tasks with jitter constraints are assigned. The earliest start time is derived by adding the offset to the period time.

In Table 8.3 it is summarized what attributes belonging to time triggered and event triggered tasks in the FPS model.

Attribute	Time triggered	Event triggered
Period	X	
MINT		X
Priority	X	X
Offset	X (Upon Jitter Constraints)	
WCET	X	X

Table 8.3: Attributes associated with time and event triggered tasks

Applying the Bate and Burns algorithm determines task attributes from the tasks and transactions described in Table 8.2. The resulting run-time attributes priority, offset period and WCET are shown in Table 8.4. The attributes offset and priority are determined with the Bate and Burns analysis, whilst the period and WCET are determined in the task allocation.

	Priority	Offset	Period	WCET
Task 1	2	0	100	15
Task 2	1 (Lowest)	35	60	5
Task 3	4 (Highest)	0	40	5
Task 4	3	0	40	15

Table 8.4: Assigned task attributes

In Figure 8.5 a run-time trace for an FPS system is shown and the transactions Tr_1 and Tr_2 are indicated.

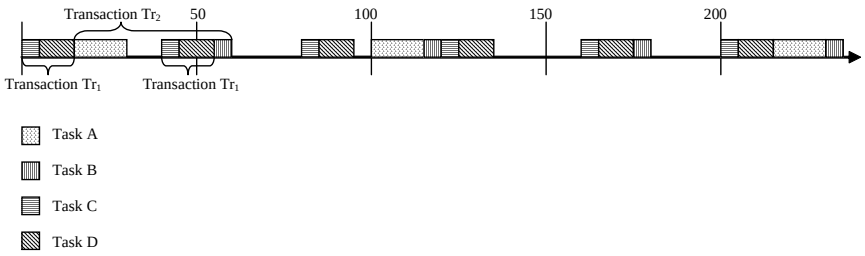

Figure 8.5: Trace of an FPS schedule

When the FPS model has been assigned its attributes it has to be verified. The verification of the model is performed by applying real-time scheduling analysis to confirm that the model is schedulable with the assigned parameters. This is necessary since attribute assignment does not necessarily guarantee schedulability, but only assigns attributes considering the relation between the tasks.

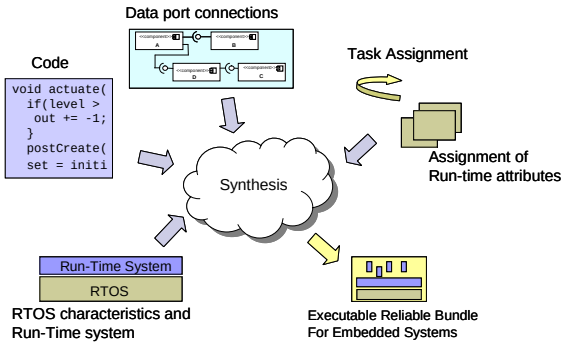

Figure 8.6: The steps of synthesizing code for the run-time system

8.4.3 Real-Time Analysis

To show that the FPS tasks will meet their stipulated timing constraints, schedulability analysis must be performed. Much research has been done with respect to analysis of different properties of FPS systems, and all those results are available for use, once a FPS model has been established. The temporal analysis of an FPS system with offsets, sporadic tasks and synchronization has been covered in research by e.g., Palencia et al. [18], [19] and Redell [20].

The output from the analysis is whether the system is feasible or not in the worst case. If the analysis shows that the system is infeasible, the parts that can not keep its requirements are either changed and reanalysed or emphasised for the developer to make changes.

8.5 Synthesis

The next step after the model transformation and real-time analysis is to synthesise code for the run-time system. This includes mapping the tasks to operating system specific task entities, mapping data connections to an OS specific communication, modifying the middleware, generating glue code, compiling, linking and bundling the program code (see Figure 8.6).

The synthesis is divided into two major parts. Given a task set and necessary information about the run-time system, the synthesis generates code considering communication, synchronization.

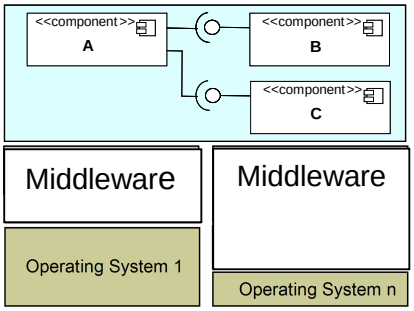

Figure 8.7: A component model with adjustments for different operating systems to promote platform independence

- The first part in synthesis is to resolve the communication within and between tasks. Two communicating components that are assigned to different tasks will form an Inter Task Communication (ITC) while communication between components assigned to the same task are realized with shared data spaces within the task. The ITC is later mapped to operating system specific communication directives.
- The other part in the synthesis is to resolve the control couplings, i.e., the sink and source. If a tasks starting point is dependent on the former tasks finishing point the tasks have to be synchronized. The synchronization is solved through scheduling. The synthesis will generate code for scheduling periodic tasks, handle the control flow between tasks and consider offsets. The code generated for the periodic scheduling and offsets is dependent on the middleware and can be realized as a configuration file or actual code in each task. Invocations of sporadic tasks are mapped to event handlers in the middleware or the operating system.

It is assumed that a middleware is present as shown in Figure 8.7, for each platform and that it provides functionality that the component model needs but the operating system does not provide. The more functionality the operating system provides, the smaller the middleware has to be. The middleware encapsulates core communication and concurrency services to eliminate many non-portable aspects of developing and is hence platform specific in favour of a platform independent component model. Typical functionality that is not provided by most commercial RTOS is periodicity and support for offsets. The

middleware also need to support sink and source couplings since task coupled with its source need to be able to invoke the corresponding task. The run-time system conforms to FPS and hence the run-time task model is similar to the previously described FPS model with some exceptions. The worst case execution time is merely an analysis attribute and is not needed in the run-time model. The MINT is usually a requirement on the environment rather than a task attribute, and is thus also analytical and unnecessary. Hence the run-time task model is for periodic tasks Period time, Priority, Offset and for sporadic tasks Priority.

8.6 Conclusions and Future Work

In this paper we show how to use component based software engineering for low footprint systems with very high demands on safe and reliable behaviour. The key concept is to provide expressive design time models and yet resource effective run-time models by statically resolve resource usage and timing by powerful compile time techniques.

The work presented in this paper introduces a component technology for resource effective and temporally verified mapping of a component model to a resource structure such as a commercial RTOS. This is made possible by introduction of a component model that support specification of high level real-time constraints, by presenting a mapping to a real-time model, permitting use of standard real-time theory, and by synthesis of run-time mechanisms for predictable execution according to the temporal specification in the component model.

Although the basic concept has been validated by successful industrial application of previous work [5], it is necessary to further validate the component technology presented here. In order to facilitate this, a prototype implementation of the component technology is under development where the core part has been completed. The prototype will enable evaluation of different technology realisations with respect to performance. Moreover, parts of the model transformation need additional attention, foremost the strategies for allocation of components to tasks. Furthermore, we will make efforts in extending the component model making it more expressive and flexible while still keeping the ability for real-time analysis. Interesting is also to investigate trade-offs between run-time foot print and flexibility with respect to e.g., adding functionality post production. Finally, the component technology will be evaluated in a larger, preferably industrial, case.

Bibliography

[1] A. Bate and I. Burns. An approach to task attribute assignment for uniprocessor systems. In *Proceedings of the 11th Euromicro Workshop on Real Time Systems, York, England*. IEEE, June 1999.

[2] K. Mok, D. Tsou, and R. C. M. De Rooij. The msp.rtl real-time scheduler synthesis tool. In *In Proc. 17th IEEE Real-Time Systems Symposium*, pages 118–128. IEEE, 1996.

[3] K. Sandström and C. Norström. Managing complex temporal requirements in real-time control systems. In *In 9th IEEE Conference on Engineering of Computer-Based Systems Sweden*. IEEE, April 2002.

[4] J. Würtz and K. Schild. Scheduling of time-triggered real-time systems. In *In Constraints, Kluwer Academic Publishers*, pages 335–357, October 2000.

[5] C. Norström, M. Gustafsson, K. Sandström, J. Mäki-Turja, and N. Bånkestad. Experiences from introducing state-of-the-art real-time techniques in the automotive industry. In *In Eigth IEEE International Conference and Workshop on the Engineering of Compute-Based Systems Washington, US*. IEEE, April 2001.

[6] Arcticus. Arcticus homepage: http://www.arcticus.se.

[7] R. van Ommering, F. van der Linden, and J. Kramer. The koala component model for consumer electronics software. In *IEEE Computer*, pages 78–85. IEEE, March 2000.

[8] G. Nierstrasz, S. Arevalo, R. Ducasse, A. Wuyts, P. Black, C. Muller, T. Zeidler, R. Genssler, and A. van den Born. A component model for

field devices. In *Proceedings of the First International IFIP/ACM Working Conference on Component Deployment, Germany*, June 2002.

[9] D. B. Stewart, R. A. Volpe, and P. K. Khosla. Design of dynamically reconfigurable real-time software using port-based objects. In *IEEE Transactions on Software Engineering*, pages 759–776. IEEE, December 1997.

[10] W. H. Schmidt and R. H. Reussner. Parameterised contracts and adaptor synthesis. In *Proc. 5th International Workshop of Component-Based Software Engineering (CBSE5)*, May 2002.

[11] S. A. Hissam, G. A. Moreno, J. Stafford, and K. C. Wallnau. Packaging predictable assem-bly with prediction-enabled component technology. Technical report, November 2001.

[12] D. K. Hammer and M. R. V. Chaudron. Component-based software engineering for resource constraint systems: What are the needs? In *6th Workshop on Object-Oriented Real-Time Dependable Systems, Rome, Italy*, January 2001.

[13] IEC. International standard IEC 1131: Programmable controllers, 1992.

[14] Mathworks. Mathworks homepage : http://www.mathworks.com.

[15] R. Yerraballi. *Scalability in Real-Time Systems.* PhD thesis, Computer Science Department, old Dominion University, August 1996.

[16] S. T. Cheng. and Agrawala A. K. Allocation and scheduling of real-time periodic tasks with relative timing constraints. In *Second International Workshop on Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 1995.

[17] R. Dobrin, G. Fohler, and P. Puschner. Translating off-line schedules into task attributes for fixed priority scheduling. In *In Real-Time Systems Symposium London, UK, December*, 2001.

[18] J. C. Palencia and M. Gonzalez Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *Proc. of the 19th Real-Time Systems Symposium*, December 1998.

[19] J. C. Palencia and M. Gonzalez Harbour. Exploiting precedence relations in the schedulabil-ity analysis of distributed real-time systems. In *Proc. of the 20th Real-Time Systems Symposium*, December 1999.

[20] O. Redell and M. Törngren. Calculating exact worst case response times for static priority scheduled tasks with offsets and jitter. In *Proc. Eighth IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE, September 2002.

Chapter 9

Paper D:
Quality Attribute Support in a Component Technology for Vehicular Software

Mikael Åkerholm, Johan Fredriksson, Kristian Sandström, and Ivica Crnkovic
In Fourth Conference on Software Engineering Research and Practice in Sweden, Linköping, Sweden, October 2004

Abstract

The electronics in vehicles represents a class of systems where quality attributes, such as safety, reliability, and resource usage, leaven all through development. Vehicular manufacturers are interested in developing their software using a component based approach, supported by a component technology, but commercial component technologies are too resource demanding, complex and unpredictable. In this paper we provide a vehicular domain specific classification of the importance of different quality attributes for software, and a discussion of how they could be facilitated by a component technology. The results can be used as guidance and evaluation for research aiming at developing component technologies suitable for vehicular systems.

9.1 Introduction

Component-based development (CBD) is of great interest to the software engineering community and has achieved considerable success in many engineering domains. CBD has been extensively used for several years in desktop environments, office applications, e-business and in general Internet- and web-based distributed applications. In many other domains, for example dependable systems, CBD is utilized to a lesser degree for a number of different reasons. An important reason is the inability of component-based technologies to deal with quality attributes as required in these domains. To identify the feasibility of the CBD approach, the main concerns of the particular domain must be identified along with how the CBD approach addresses these concerns and what is its ability to provide support for solutions related to these concerns are.

There is currently a lot of research on predicting and maintaining different quality attributes within the Component Based Software Engineering (CBSE) community, (also called non-functional properties, extra-functional properties, and illities), [1] [2] [3], [4], [5]. Many of the quality attributes are conflicting and cannot be fully supported at the same time [6], [7]. Thus, it is important for application and system developers to be able to prioritize among different quality attributes when resolving conflicts.

We provide a domain specific classification of the importance of quality attributes for software in vehicles, and discuss how the attributes could be facilitated by a component technology. The discussion contribute with a general description of the desired quality attribute support in a component technology suitable for the vehicle domain and it indicates which quality attributes require explicit support. In addition, it discusses were in the technology the support should be implemented: inside or outside the components, in the component framework, on the system architecture level, or if the quality attributes are usage dependent. Quality attributes might be conflicting; e.g., it is commonly understood that flexibility and predictability are conflicting. The ranking provided by industrial partners gives domain specific guidance for how conflicts between quality attributes should be resolved. The results also enable validation and guidance for future work regarding quality attribute support in component technologies for software in vehicular systems. This guideline can be used to verify that the right qualities are addressed in the development process and that conflicting interdependent quality attributes are resolved according to the domain specific priorities.

The starting point of this work is a list of quality attributes ranked according to their importance for vehicular systems. The list is provided through a

set of interviews and discussions with experts from different companies in the vehicular domain. The results of the ranking from the vehicular companies are combined with the classification of how to support different quality attributes provided in [8]. The result is an abstract description of where, which, and how different quality attributes should be supported by a component technology tailored for the vehicular industry.

A component technology as defined in [9] is a technology that can be used for building component based software applications. It implements a component model defining the set of component types, their interfaces, and, additionally, a specification of the allowable patterns of interaction among component types. A component framework is also part of the component technology, its role can be compared to the role of an operating system, and it provides a variety of deployment and run-time services to support the component model. Specialized component technologies used in different domains of embedded systems have recently been developed, e.g., [10, 11]. There are also a number of such component technologies under development in the research community, e.g., [12, 13, 14]. The existence of different component technologies can be motivated by their support for different quality attributes, although they follow the same CBSE basic principles. It has been shown that companies developing embedded systems in general consider different non functional quality attributes far more important than efficiency in software development, which explains the specialization of component technologies [12].

The outline of the remaining part of the paper is as follows. Section 9.2 describes the conducted research method, and section 9.3 the results. Section 9.4 is a discussion of the implications of the results, regarding the support for quality attributes in a domain specific component technology. Section 9.5 discusses future work, and finally the section 9.6 concludes the paper.

9.2 Method

The research method is divided into three ordered steps:

1. During the first step a list of relevant quality attributes was gathered;

2. In the next step technical representatives from a number of vehicular companies placed priorities on each of the attributes in the list reflecting their companies view respectively;

3. Finally a synthesis step was performed, resulting in a description of the

desired quality attribute support in a component technology for vehicular systems.

The list of quality attributes have been collected from different literature trying to cover qualities of software that interest vehicular manufactures. In order to reduce a rather long list, attributes with clear similarities in their definitions have been grouped in more generic types of properties, e.g., portability and scalability are considered covered by maintainability. Although such grouping could fade the specific characteristics of a particular attribute, it put focus on the main concerns. In the ISO 9126 standard [15], 6 quality attributes (functionality, reliability, usability, efficiency, maintainability, and portability) are defined for evaluation of software quality. However, the standard has not been adopted fully in this work; it is considered too brief and does not cover attributes important for embedded systems (e.g., safety, and predictability). Furthermore, concepts that sometimes are mixed with quality attributes (for example fault tolerance) are not classified as quality attributes, rather as methods to achieve qualities (as for example safety). Finally, functionality is of course one of the most important quality attributes of a product, indicating how well it satisfies stated or implied needs. However, we focus on quality attributes beyond functionality often called extra-functional or non-functional properties. The resulting list of quality attributes is presented below.

Extendibility the ease with which a system or component can be modified to increase its storage or functional capacity.

Maintainability the ease with which a software system or component can be modified to correct faults, improve performance, or other attributes, or adapt to a changed environment.

Usability the ease with which a user can learn to operate, prepare inputs for, and interpret outputs from a system or component.

Predictability to which extent different run-time attributes can be predicted during design time.

Security the ability of a system to manage, protect, and distribute sensitive information.

Safety a measure of the absence of unsafe software conditions. The absence of catastrophic consequences to the environment.

Reliability the ability of a system or component to perform its required functions under stated conditions for a specified period of time.

Testability the degree to which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met. Note: testability is not only a measurement for software, but it can also apply to the testing scheme.

Flexibility the ease with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed.

Efficiency the degree to which a system or component performs its designated functions with minimum consumption of resources (CPU, Memory, I/O, Peripherals, Networks).

Representatives from the technical staff of several companies have been requested to prioritize a list of quality attributes, reflecting each of the respective companiesŠ view. The attributes have been grouped by the company representatives in four priority classes as shown in Table 9.1. The nature of the quality attributes imply that no quality attribute can be neglected. It is essential to notice that placing an attribute in the lowest priority class (4) does not mean that the company could avoid that quality in their software, rather that the company does not spend extra efforts in reaching it. The following companies have been involved in the classification process:

- Volvo Construction Equipment [16] develops and manufactures a wide variety of construction equipment vehicles, such as articulated haulers, excavators, graders, backhoe loaders, and wheel loaders.
- Volvo Cars [17] develops passenger cars in the premium segment. Cars are typically manufactured in volumes in the order of several hundred thousands per year.
- Bombardier Transportation [18] is a train manufacturer, with a wide range of related products. Some samples from their product line are passenger rail vehicles, total transit systems, locomotives, freight cars, propulsion and controls, and signalling equipment.
- Scania [19] is a manufacturer of heavy trucks and buses as well as industrial and marine engines.
- ABB Robotics [20] is included in the work as a reference company, not acting in the vehicular domain. They are building industrial robots, and it is the department developing the control systems that is represented.

Priority	Description
1	very important, must be considered
2	important, something that one should try to consider
3	less important, considered if it can be achieved with a small effort
4	Unimportant, do not spend extra effort on this

Table 9.1: Priority classes used to classify the importance of the different quality attributes

As the last step we provide a discussion where we have combined the collected data from the companies with the classification of how to support different quality attributes in [8]. The combination gives an abstract description of where, which, and how different quality attributes should be supported by a component technology tailored for usage in the vehicular industry.

9.3 Results

Figure 9.1 is a diagram that summarizes the results. The attributes are prioritized by the different companies, in a scale from priority 1 (highest), to 4 (lowest) indicated on the Y-axis. On the X-axis the attributes are presented with the highest prioritized attribute as the leftmost, and lowest as rightmost. Each of the companies has one bar for each attribute, textured as indicated below the X-axis. In some cases the representatives placed an interval for the priority of certain attributes, e.g., 1-3 dependent on application; in those cases the highest priority has been chosen in the diagram.

The result shows that the involved companies have approximately similar prioritization, except on the security quality attribute where we have both highest and lowest priority. Reasonably, the most important concerns are related to dependability characteristics (i.e. to the expectation of the performance of the systems): safety, reliability and predictability. Usability is a property important for the customers but also crucial in competition on the market. Slightly less important attributes are related to the life cycle (extendibility, maintainability). This indicates that the companies are ready to pay more attention to the product performance than to the development and production costs (in that sense a component-based approach which primary concerns are of business nature, might not necessary be the most desirable approach).

The results also shows that ABB Robotics, included as a reference com-

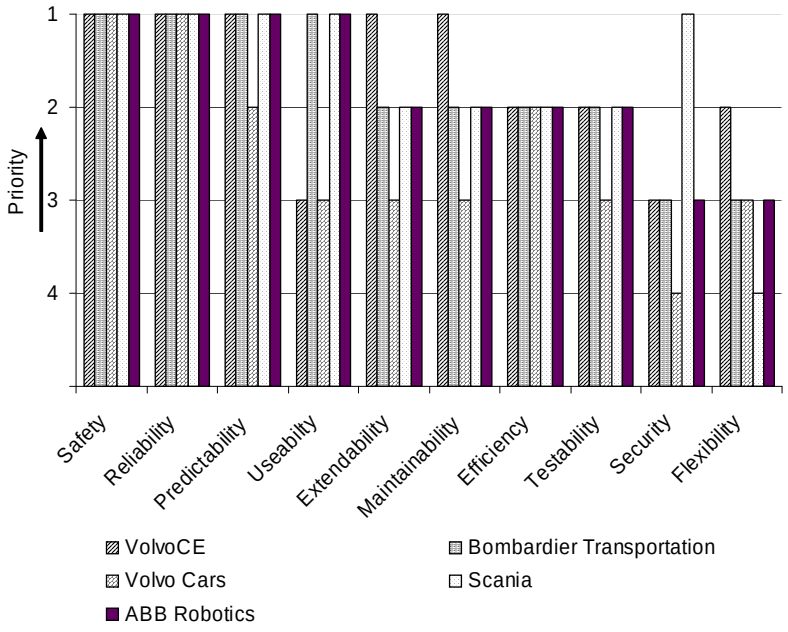

Figure 9.1: the results. Y-axis: priority of quality attributes in a scale 1 (highest), to 4 (lowest). X-axis: the attributes, with the highest prioritized attribute as the leftmost, and lowest as rightmost. Each of the companies has one bar for each attribute, textured as indicated below the X-axis.

pany outside the vehicular domain has also approximately the same opinion. It is not possible to distinguish ABB Robotics from any of the vehicular companies from a quality attribute perspective. These companies might use the same component technology with respect to quality attribute support; thus the results in the investigation indicate that the priority among quality attributes scale to a broader scope of embedded computer control systems.

9.4 Discussion of the results

A component technology may have built in support for maintaining quality attributes. However, tradeoffs between quality attributes must be made since they are interdependent [7, 6]. We will discuss how the different quality attributes can be supported by a component technology, and suggest how necessary tradeoffs can be made according to priority placed by industry. The discussion starts by treating the attribute that has received the highest priority (safety), and continues in priority order, in this way the conflicts (and tradeoffs) will be discussed in priority order. As basis for where support for a specific quality attribute should be implemented we use a classification from [8], listed below:

- Directly composable, possible to analyze given the same quality attributes from the components.
- Architecture related, possible to analyze given this attribute for the components and the assembly architecture.
- Derived attributes, possible to analyze from several attributes from the involved components.
- Usage dependent, need a usage profile to analyze this.
- System environment context, possible to analyze given environment attributes.

9.4.1 Safety

Safety is classified as dependent on the usage profile, and the system environment context. Similarly to the fact that we cannot reason about system safety without taking into consideration the surrounding context, we cannot reason about safety of a component: simply safety is not a property that can be identified on the component level. But a component technology can include numerous mechanisms that enhance safety, or simplify safety analysis. However, to perform safety analysis, usage and environment information is needed. A component technology can have support for safety kernels [21], surrounding components and supervise that unsafe conditions do not occur. Pre- and post conditions can be checked in conjunction with execution of components to detect hazardous states and check the range of input and output, used in specification of components in e.g., [22, 23]. Tools supporting safety analysis

as fault tree analysis (FTA) or failure modes and effect analysis (FMEA) can also be provided with the component technology.

9.4.2 Reliability

Reliability is architecture related and usage dependent. The dominant type of impact on reliability is the usage profile but reliability is also dependent on the software architecture and how components are assembled; a fault-tolerant redundant architecture improves the reliability of the assembly of components. One possible approach to calculation of the reliability of an assembly is to use the following elements:

- Reliability of the components - Information that has been obtained by testing and analysis of the component given a context and usage profile.
- Path information (usage paths) - Information that includes usage profile and the assembly structure.

Combined, it can give a probability of execution of each component, for example by using Markov chains.

Also common for many simple systems, the reliability for a function of two components is calculated using the reliability of the components, and their relationship when performing the function. An AND relationship is when the output is dependent on correct operation of both components, and an OR occurs when the output is created when one of the two components operates correctly.

A component technology could have support for reliability, through reliability attributes associated with components, and tools that automatically determines reliability of given usage profiles, path information, and structural relationships.

It is noteworthy that even if the reliability of the components are known it is very hard to know if side effects take place that will affect an assembly of the components. E.g. a failure caused by a component writing in a memory space used by another component. A model based on these assumptions needs the means for calculating or measuring component reliability and an architecture that permits analysis of the execution path. Component models that specify provided and required interface, or implement a port-based interface make it possible to develop a model for specifying the usage paths. This is an example in which the definition of the component model facilitates the procedure of dealing with the quality attribute. One known problem in the use of Markov chains in modelling usage is the rapid growth of the chain and complexity

[24]. The problem can be solved because the reliability permits a hierarchical approach. The system reliability can be analyzed by (re)using the reliability information of the assemblies and components (which can be derived or measured).

Reliability and Safety are not conflicting attributes. Reliability enhances safety, high reliability increases confidence that the system does what it is intended to and nothing else that might lead to unsafe conditions.

9.4.3 Predictability

We focus on predictability of the particular run-time attributes temporal behaviour, memory consumption, and functional behaviour. Predictability is directly composable and architecture dependent. Prediction of temporal behaviour is well explored in research within the real-time community. Depending on the run-time systems scheduling strategy, the shared resource access and execution demands of the scheduled entities, suitable prediction theories can be chosen, e.g., for fixed priority systems that are most common within industry [25, 26]. The choice of scheduling strategy is also a problem that has been addressed [27]. Static scheduled systems are more straightforward to predict than event driven systems that on the other hand are more flexible. Memory consumption can be predicted, given the memory consumption for the different components in the system [28]. However, two different types of memory consumption can be identified: static and dynamic. Static memory consumption is the most straightforward to predict, since it is a simple summation of the memory requirements of the included components. Dynamic memory consumption can be more complex, since it might be dependent on usage input, and thereby be usage dependent.

Predictability is not in conflict with the higher prioritized attributes reliability and safety. Predictable behaviour enhances safety and reliability, e.g., unpredictable behaviour cannot be safe because it is impossible to be sure that certain actions will not take place.

9.4.4 Usability

Usability is a rather complex quality attribute, which is derived from several other attributes; it is architecture related and usage dependent. Usability is not directly related to selection of component technology. Software in embedded systems (the most common and important type of software in vehicular systems) is usually not visible and does not directly interact with the user. How-

ever, more and more human-machine interaction is implemented in underlying software. In many cases we can see how the flexibility of software is abused - there are many devices (for example in infotainment) with numerous buttons and flashing screens that significantly decrease the level of usability. Use of a component technology may however indirectly contribute to usability - by building standard (user-interface) components, and by their use in different applications and products, the same style, type of interaction, functionality and similar are repeated. In this way they become recognisable and consequently easier to use.

Usability as discussed above is not in obvious conflict with any of the higher prioritized quality attributes.

9.4.5 Extendibility

Extendibility is directly composable and architecture related. It can be supported by the component technology through absence of restrictions in size related parameters, e.g., memory size, code size, and interface size. Extendibility is one of the main concerns of a component technology and it is explicitly supported either by ability of adding or extending interfaces or by providing a framework that supports extendibility by easy updating of the system with new or modified components.

Extendibility is not in direct conflict with any of the higher prioritized attributes. However, conflicts may arise due to current methods used for analysis and design of safety critical systems real-time systems, the methods often results in systems that are hard to extend [29]. Predictability in turn enhances extendibility, since it makes predications of the impact of an extension possible.

9.4.6 Maintainability

Maintainability is directly composable and architecture related. A component technology supports maintainability through configuration management tools, clear architectures, and possibilities to predict impacts of applied changes.

Maintainability is not in obvious conflict with any of the higher prioritized attributes. But as for extendibility, current state of practice for achieving safety, dependability and predictability results in systems that often are hard to maintain [29]. Maintainability increases usability, while good predictability in turn increases maintainability since impacts of maintenance efforts can be predicted.

9.4.7 Efficiency

Efficiency is directly composable and architecture related. Efficiency is affected by the component technology, mainly through resource usage by the run-time system but also by interaction mechanisms. Good efficiency is equal to low memory, processor, and communication medium usage.

In the requirements for a software application it might often be the case that a certain amount of efficiency is a basic requirement, because of limited hardware resources, control performance, or user experienced responsiveness. In such cases the certain metrics must be achieved, but efficiency is potentially in conflict with many higher prioritized quality attributes. Safety related run-time mechanisms as safety kernels, and checking pre- and post conditions consume extra resources and are thus in conflict with efficiency. Reliability is often increased by redundancy, by definition conflicting with efficiency. Methods used for guaranteeing real-time behaviour are pessimistic and result in low utilization bounds [30], although it is a widely addressed research problem and improvements exist, e.g., [31, 32].

9.4.8 Testability

Testability is directly composable and architecture related. A general rule for testability is that simple systems are easier to test than complex systems; however, what engineers build is not directly related to the technology itself. Direct methods to increase testability provided by a component technology can be built in self tests in components, monitoring support in the run-time system, simulation environments, high and low level debugging information [33].

Testability is not in conflict with any of the higher prioritized quality attributes. On the contrary, it supports several other attributes, e.g., safety is increased by testing that certain conditions cannot occur, predictions are confirmed by testing, maintainability is increased if it is possible to test the impact of a change. However, efficiency tradeoffs might have to be done to enable testing. A problem with many common testing methods is the probe effect introduced by software probes used for observing the system [34]. If the probes used during testing are removed in the final product, it is not the same system that is delivered as the one tested. To avoid this problem, designers can choose to leave the probes in the final product and sacrifice efficiency, or possibly use some form of non-intrusive hardware probing methods, e.g., [35]. Reliability implemented by fault tolerance decrease testability, since faults may become hidden and complicate detection by testing.

9.4.9 Security

Security is usage dependent and dependent on the environment context, meaning that it is not directly affected by the component technology. However, mechanisms increasing security can be built in a component technology, e.g., encryption of all messages, authorization of devices that communicate on the bus.

Methods to increase security that can be built in a component technology are often in conflict with higher prioritized quality attributes, e.g., encryption is in conflict with efficiency since it require more computing, and with testability since it is harder to observe the system. Furthermore security has a low priority, and the methods to achieve it are not dependent on support from the component technology. Hence, security can be implemented without support from the component technology.

9.4.10 Flexibility

Flexibility is directly composable and architecture related. A component technology can support flexibility through the components, their interactions, and architectural styles to compose systems. Methods increasing flexibility in a component technology can be, e.g., dynamic run-time scheduling of activities based on events, run-time binding of resources, and component reconfiguration during run-time.

Flexibility has received the lowest priority of all quality attributes, and is in conflict with many higher prioritized attributes, e.g., with safety since the number of different hazardous conditions increases, with testability since the number of test cases increases and it may not be possible at all to create a realistic run-time situation thus not to test the actual system either. On the other hand flexibility increases maintainability, since a flexible system is easier to change during maintenance. It is not possible to use completely static systems with no flexibility at all when user interaction is involved, but regarding to the numerous conflicts with higher prioritized quality attributes it should be kept to a minimum in component technologies for this domain.

9.4.11 Quality Attribute Support in a Component Technology for the Automotive Domain

Having presenting the basic characteristics of quality attributes related to component technologies, and identification of present conflicts, and suggestions on

how to resolve the conflict we give a brief description of the resulting suggestion of support for quality attributes in a component technology tailored for vehicular systems below:

Safety Safety cannot be fully supported by a component technology. However, safety kernels surrounding components and support for defining pre- and post conditions are suggested.

Reliability Reliability is supported to a large extent by a component technology. We suggest reliability attributes associated with components, path information including usage profile and assembly structure, and tools for analysis. There should also be support for redundant components when necessary.

Predictability Predictability is supported to a large extent. Associated to the components, attributes such as execution time, and memory consumption can be specified. Tools for automated analysis can be provided with the technology.

Usability Usability is not directly supported by a component technology.

Extendibility Extendibility is well supported. The interfaces should be easy to extend and it should be easy to add new components to an existing system. There should be no size related restrictions with respect to memory, code, and interface.

Maintainability Maintainability is well supported by a component technology. The support is provided through configuration management tools, and the fact that using well defined components gives a clear and maintainable architecture.

Efficiency Efficiency is suggested to be supported to a fairly high level. We suggest support through small and efficient run-time systems, however not to the cost of suggested safety and reliability related run-time mechanisms.

Testability Testability is supported to a large extent. The support is suggested to be monitoring possibilities in the run-time system, simulation and debug possibilities.

Security Security is not directly supported.

Flexibility Flexibility is not directly supported.

9.5 Future Work

We will continue with research towards enabling CBSE for automotive systems. One part is to continue investigating the requirements on quality attributes from the domain, with our present and other industrial partners. Another part is an analysis of particular component models to investigate their abilities of supporting these quality attributes. A third part is to enable support for quality attributes in the component technologies we are developing as prototypes suitable for the domain AutoComp [36] and SaveComp [12], but we will also asses to which extent other existing component technologies can be used in order to meet the industrial requirements.

9.6 Conclusions

We have presented a classification of the importance of quality attributes for software made by some companies in the vehicular domain; the results showed that the companies agreed upon the priority for most of the attributes. The most important concerns showed to be related to dependability characteristics (safety, reliability and predictability). Usability received a fairly high priority. Slightly less important attributes where those related to the life cycle (extendibility, maintainability), while security and flexibility received the lowest priority. We also included a company outside the domain in the investigation, it turned out that they also agreed upon the classification; it might be that the classification scale to a broader scope of embedded systems.

Furthermore, we have discussed how the attributes could be facilitated by a component technology, and were in the technology the support should be implemented: inside or outside the components, in the framework, or if the quality attributes are usage dependent. The discussion is concluded by a brief suggestion of quality attribute support for a component technology.

Acknowledgements We would like to thank our industrial partners for their time, and the interest they have shown in discussing quality attributes. Thanks to Joakim Fröberg from Volvo CE, Jakob Axelsson from Volvo Cars, Mattias Ekman from Bombardier Transportation, Ola Larses and Bo Neidenström from Scania, and Bertil Emertz from ABB Robotics.

Bibliography

[1] I. Crnkovic and M. Larsson. Classification of quality attributes for predictability in component-based systems. In *In DSN 2004 Workshop on Architecting Dependable Systems*, June 2004.

[2] G.A. Moreno, S.A. Hissam, and K.C. Wallnau. Statistical models for empirical component properties and assembly-level property predictions: Towards standard labeling. In *Proceedings of 5th Workshop on component based software engineering*, 2002.

[3] R.H. Reussner, H.W. Schmidt, and I. Poernomo. Reliability prediction for component-based software architectures. *Journal of Systems and Software*, 66(3):241–252, 2003.

[4] H.W. Schmidt. Trustworthy components: Compositionality and prediction. *Journal of Systems and Software*, 65(3):212–225, 2003.

[5] J. Stafford and J. McGregor. Issues in the reliability of composed components. In *5th workshop on component based software engineering (CBSE5)*, 2002.

[6] D. Haggander, L. Lundberg, and J. Matton. Quality attribute conflicts - experiences from a large telecommunication application. In *Proceedings of the 7th IEEE International Conference on Engineering of Complex Computer Systems*, 2001.

[7] M. Barbacci, M. H. Klein, T. A. Longstaff, and C. B. Weinstock. Quality attributes. Technical report, Software Engineering Institute, Carnegie Mellon University, 1995.

[8] M. Larsson. *Predicting Quality Attributes in Component-based Software Systems*. PhD thesis, Mälardalen University, March 2004.

[9] F. Bachmann, L. Bass, C. Buhman, S. Comella-Dorda, F. Long, J. Robert, R. Seacord, and K. Wallnau. Technical concepts of component-based software engineering, volume ii. Technical report, Software Engineering Institute, Carnegie-Mellon University, May 2000. CMU/SEI-2000-TR-008.

[10] O. Nierstrass, G. Arevalo, S. Ducasse, , R. Wuyts, A. Black, P. Müller, C. Zeidler, T. Genssler, and R. van den Born. A Component Model for Field Devices. In *Proceedings of the First International IFIP/ACM Working Conference on Component Deployment*, June 2002.

[11] R. van Ommering, F. van der Linden, and J. Kramer. The koala component model for consumer electronics software. In *IEEE Computer*, pages 78–85. IEEE, March 2000.

[12] H. Hansson, M. Åkerholm, I. Crnkovic, and M. Törngren. SaveCCM - a Component Model for Safety-Critical Real-Time Systems. In *Proceedings of 30th Euromicro Conference, Special Session Component Models for Dependable Systems*, September 2004.

[13] M. de Jonge, J. Muskens, and M. Chaudron. Scenario-based prediction of run-time resource consumption in component-based software systems. In *Proceedings of the 6th ICSE Workshop on Component-Based Software Engineering (CBSE6)*, May 2003.

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

[15] ISO/IEC. *Software engineering – Product quality – Part 1: Quality model, ISO/IEC 9126-1*, 2001.

[16] Volvo construction equipment homepage. http://www.volvo.com/constructionequipment.

[17] Volvo cars homepage. http://www.volvocars.com/.

[18] Bombardier transportation homepage. http://www.transportation.bombardier.com/.

[19] Scania homepage. http://www.scania.com/.

[20] Abb robotics homepage. http://www.abb.com/robotics.

[21] J. Rushby. Kernel for safety, in safe and secure computing systems. Technical report, Blackwell Scientific Publications, Londres, 1989.

[22] J. Chessman and J. Daniels. *UML Componets - A simple process for specifying Component-Based Software*. Reading, MA: Addison-Wesley, 2000.

[23] D. D'Souza and A.C. Wills. *Objects, Components and Frameworks: The Catalysis Approach*. Reading, MA: Addison-Wesley, 1998.

[24] H.W. Schmidt and R.H. Reussner. Parameterized Contracts and Adapter Synthesis. In *Proceedings of the 5th ICSE Workshop on Component-Based Software Engineering*, May 2001.

[25] N. C. Audsley, A. Burns, R. I. Davis, K. W. Tindell, , and A. J. Wellings. Fixed priority pre-emptive scheduling: An historical perspective. *Real-Time Systems Journal*, 8(2/3):173–198, 1995.

[26] O. Redell and M. Törngren. Calculating exact worst case response times for static priority scheduled tasks with offsets and jitter. In *Proc. Eighth IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE, September 2002.

[27] J. Xu and D. L. Parnas. On Satisfying Timing Constraints in Hard-Real-Time Systems. *IEEE Transactions on Software Engineerin*, 19(1):70–84, 1993.

[28] A.V. Fioukov, E.M. Eskenazi, D.K. Hammer, and M. Chaudron. Evaluation of Static Properties for Component-Based Architetures. In *Proceedings of 28th Euromicro Conference*, September 2002.

[29] A. Burns and J. A. McDermid. Real-time safety-critical systems: analysis and synthesis. *Software Engineering Journal*, 9(6):267–281, Nov 1994.

[30] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in hard-real-time environment. *Journal of the Association for Computing Machinery (ACM)*, 20(1):46–61, 1973.

[31] T. F. Abdelzaher, V. Sharma, and C. Lu. A utilization bound for aperiodic tasks and priority driven scheduling. *IEEE Transactions on Computers*, 53(3):334–350, Mar 2004.

[32] C. Deji, A. K. Mok, and K. Tei-Wei. Utilization bound revisited. *IEEE Transactions on Computers*, 52(3):351–361, March 2003.

[33] H. Thane. *Testing and Debugging of Distributed Real-Time Systems*. PhD thesis, Royal Institute of Technology, May 2000.

[34] J. Gait. A probe effect in concurrent programs. *Software Practise and Experience*, 16(3), March 1986.

[35] M. El Shobaki and L. Lindh. A hardware and software monitor for high-level system-on-chip verification. In *Proceedings of the IEEE International Symposium on Quality Electronic Design*, March 2001.

[36] K. Sandström, J. Fredriksson, and M. Åkerholm. Introducing a Component Technology for Safety Critical Embedded Real-Time Systems. In *Proceedings of th 7th International Symposium on Component-Based Software Engineering (CBSE7)*, May 2004.

Chapter 10

Paper E: Evaluation of Component Technologies with Respect to Industrial Requirements

Anders Möller, Mikael Åkerholm, Johan Fredriksson, and Mikael Nolin
In Euromicro Conference, Component-Based Software Engineering Track Rennes, France, August 2004

Abstract

We compare existing component technologies for embedded systems with respect to industrial requirements. The requirements are collected from the vehicular industry, but our findings are applicable to similar industries developing resource constrained safety critical embedded distributed real-time computer systems.

One of our conclusions is that none of the studied technologies is a perfect match for the industrial requirements. Furthermore, no single technology stands out as being a significantly better choice than the others; each technology has its own pros and cons.

The results of our evaluation can be used to guide modifications or extensions to existing technologies, making them better suited for industrial deployment. Companies that want to make use of component-based software engineering as available today can use this evaluation to select a suitable technology.

10.1 Introduction

Component-Based Software Engineering (CBSE) has received much attention during the last couple of years. However, in the embedded-system domain, use of component technologies has had a hard time gaining acceptance; software-developers are still, to a large extent, using monolithic and platform-dependent software technologies.

We try to find out why embedded-software developers have not embraced CBSE as an attractive tool for software development. We do this by evaluating a set of component technologies with respect to industrial requirements. The requirements have been collected from industrial actors within the business segment of heavy vehicles, and have been presented in our previous work [1]. Examples of heavy vehicles include wheel loaders, excavators, forest harvesters, and combat vehicles. The software systems developed within this market segment can be characterised as resource constrained, safety critical, embedded, distributed, real-time, control systems. Our findings in this study should be applicable to other domains with similar characteristics.

Our evaluation, between requirements and existing technologies, does not only help to answer why component-based development has not yet been embraced by the embedded-systems community. It also helps us to identify what parts of existing technologies could be enhanced, to make them more appropriate for embedded-system developers. Specifically, it will allow us to select a component technology that is a close match to the requirements, and if needed, guide modifications to that technology.

The reason for studying component-based development in the first place, is that software developers can achieve considerable business benefits in terms of reduced costs, shortened time-to-market and increased software quality by applying a suitable component technology. The component technology should rely on powerful design and compile-time mechanisms and simple and predictable run-time behaviour.

There is however significant risks and costs associated with the adoption of a new development technique (such as component-based development). These must be carefully evaluated before introduced in the development process. One of the apparent risks is that the selected component technology turns out to be inappropriate for its purpose; hence, the need to evaluate component technologies with respect to requirements expressed by software developers.

10.2 Requirements

The requirements discussed and described in this section are based on a previously conducted investigation [1]. The requirements found in that investigation are divided into two main groups, the technical requirements (Section 10.2.1) and the development process related requirements (Section 10.2.2). In addition, Section 10.2.3 contains derived requirements, i.e. requirements that we have synthesised from the requirements in Sections 10.2.1 and 10.2.2 but that are not explicitly stated requirements from the vehicular industry [1].

10.2.1 Technical Requirements

The technical requirements describe industrial needs and desires regarding technical aspects and properties of a component technology.

Analysable

System analysis, with respect to non-functional properties, such as timing behaviour and memory consumption is considered highly attractive. In fact, it is one of the single most distinguished requirements found in our investigation.

When analysing a system built from well-tested, functionally correct, components, the main issue is associated with composability. The composition process must ensure that non-functional properties, such as the communication, synchronisation, memory, and timing characteristics of the system, are predictable [2].

Testable and debugable

It is required that tools exist that support debugging, both at component level (e.g., a graphical debugging tool), as well as on source code level.

Testing and debugging is one of the most commonly used techniques to verify software systems functionality. Testing is a very important complement to analysis, and testability should not be compromised when introducing a component technology. Ideally, the ability to test embedded-system software should be improved when using CBSE, since it adds the ability to test components in isolation.

Portable

The components, and the infrastructure surrounding them, should be platform independent to the highest degree possible. Here, platform independency means (1) hardware independent, (2) real-time operating system (RTOS) independent and (3) communications protocol independent. The components are kept portable by minimising the number of dependencies to the software platform. Eventually such dependencies are off course necessary to construct an executable system, however the dependencies should be kept to a minimum, and whenever possible dependencies should be generated automatically by configuration tools.

Resource Constrained

The components should be small and light-weighted and the components infrastructure and framework should be minimised. Ideally there should be no run-time overhead compared to not using a CBSE approach. Hardware used in embedded real-time systems is usually resource constrained, to lower production cost and thereby increase profit.

One possibility, that significantly can reduce resource consumption of components and the component framework, is to limit run-time dynamics. This means that it is desirable only to allow static, off-line, configured systems. Many existing component technologies have been design to support high run-time dynamics, where components are added, removed and reconfigured during run-time.

Component Modelling

The component modelling should be based on a standard modelling language like UML [3] or UML 2.0 [4]. The main reason to choose a standard like UML is that it is well known and thoroughly tested, with tools and formats supported by many third-party developers. The reason for the vehicular industry to have specific demands in this detail, is that this business segment does not have the knowledge, resources or desire to develop their own standards and practices.

Computational Model

Components should preferably be passive, i.e., they should not contain their own threads of execution. A view where components are allocated to threads

during component assembly is preferred, since this is conceptually simple, and also believed to enhance reusability.

The computational model should be focused on a pipes-and-filters model [5]. This is partly due to the well known ability to schedule and analyse this model off-line. Also, the pipes-and-filters model is a good conceptual model for control applications.

10.2.2 Development Requirements

When discussing component-based development with industry, development process requirements are at least as important as the technical requirements. To obtain industrial reliance, the development requirements need to be addressed by the component technology and its associated tools.

Introducible

Appropriate support to gradually migrate to a new technology should be provided by the component technology. It is important to make the change in development process and techniques as safe and inexpensive as possible. Revolutionary changes in development techniques are associated with high risks and costs. Therefore a new technology should be possible to divide into smaller parts, which can be introduced incrementally. Another aspect, to make a technology introducible, is to allow legacy code within systems designed with the new technology.

Reusable

Components should be reusable, e.g., for use in new applications or environments than those for which they where originally designed [6]. Reusability can more easily be achieved if a loosely coupled component technology is used, i.e., the components are focusing on functionality and do not contain any direct operating system or hardware dependencies. Reusability is further enhanced by the possibility to use configuration parameters to components.

A clear, explicit, and well-defined component interface is crucial to enhance the software reusability. Also, specification of non-functional properties and requirements (such as execution time, memory usage, deadlines, etc.) simplify reuse of components since it makes (otherwise) implicit assumptions explicit. Behavioural descriptions (such as state diagrams or interaction diagrams) of components can be used to further enhance reusability.

Maintainable

The components should be easy to change and maintain, developers that are about to change a component need to understand the full impact of the proposed change. Thus, not only knowledge about component interfaces and their expected behaviour is needed. Also, information about current deployment contexts may be needed in order not to break existing systems. The components can be stored in a repository where different versions and variants need to be managed in a sufficient way. The maintainability requirement also includes sufficient tools supporting the service of deployed and delivered products. These tools need to be component aware and handle error diagnostics from components and support for updating software components.

Understandable

The component technology and the systems constructed using it should be easy to understand. This should also include making the technology easy and intuitive to use in a development project.

The reason for this requirement is to simplify evaluation and verification both on the system level and on the component level. Focusing on an understandable model makes the development process faster and it is likely that there will be fewer bugs. This requirement is also related to the introducible requirement (Section 10.2.2) since an understandable technique is more introducible.

It is desirable to hide as much complexity as possible from system developers. Ideally, complex tasks (such as mapping signals to memory areas or bus messages, or producing schedules or timing analysis) should be performed by tools.

10.2.3 Derived Requirements

Here, we present requirements that we have synthesised from the requirements in sections 10.2.1 and 10.2.2, but that are not explicit requirements from industry.

Source Code Components

A component should be source code, i.e., no binaries. Companies are used to have access to the source code, to find functional errors, and enable support for white box testing (Section 10.2.1). Since source code debugging is demanded, even if a component technology is used, black box components is undesirable.

However, the desire to look into the components does not necessary imply a desire to be allowed to modify them[1].

Using black-box components would lead to a fear of loosing control over the system behaviour (Section 10.2.2). Provided that all components in the systems are well tested, and that the source code are checked, verified, and qualified for use in the specific surrounding, the companies might alleviate their source code availability.

Also with respect to the resource constrained requirement (Section 10.2.1), source code components allow for unused parts of the component to be removed at compile time.

Static Configurations

Better support for the technical requirements of analysability (Section 10.2.1), testability (Section 10.2.1), and resource consumption (Section 10.2.1), are achieved by using pre-runtime configuration. Here, configuration means both configuration of component behaviour and interconnections between components. Component technologies for use in the Office/Internet domain usually focus on dynamic configurations [7, 8]. This is of course appropriate in these specific domains, where one usually has access to ample resources. Embedded systems, however, face another reality; with resource constrained nodes running complex, dependable, control applications.

However, most vehicles can operate in different modes, hence the technology must support switches between a set of statically configured modes. Static configuration also improves the development process related requirement of understandability (Section 10.2.2), since each possible configuration is known before run-time.

10.3 Component Technologies

In this section, existing component technologies for embedded systems are described and evaluated. The technologies originate both from academia and industry. The selection criterion for a component technology has firstly been that there is enough information available, secondly that the authors claim that

[1] This can be viewed as a "glass box" component model, where it possible to acquire a "use-only" license from a third party. This license model is today quite common in the embedded systems market.

the technology is suitable for embedded systems, and finally we have tried to achieve a combination of both academic and industrial technologies.

The technologies described and evaluated are PECT, Koala, Rubus Component Model, PBO, PECOS and CORBA-CCM. We have chosen CORBA-CCM to represent the set of technologies existing in the PC/Internet domain (other examples are COM, .NET [7] and Java Enterprise Beans [8]) since it is the only technology that explicitly address embedded and real-time issues. Also, the Windows CE version of .NET [7] is omitted, since it is targeted towards embedded display-devices, which only constitute a small subset of the devices in vehicular systems. The evaluation is based on publicly available, documentation.

10.3.1 PECT

A Prediction-Enabled Component Technology (PECT) [9] is a development infrastructure that incorporates development tools and analysis techniques. PECT is an ongoing research project at the Software Engineering Institute (SEI) at the Carnegie Mellon University.[2] The project focuses on analysis; however, the framework does not include any concrete theories - rather definitions of how analysis should be applied. To be able to analyse systems using PECT, proper analysis theories must be found and implemented and a suitable underlying component technology must be chosen.

A PECT include an abstract model of a component technology, consisting of a construction framework and a reasoning framework. To concretise a PECT, it is necessary to choose an underlying component technology, define restrictions on that technology (to allow predictions), and find and implement proper analysis theories. The PECT concept is highly portable, since it does not include any parts that are bound to a specific platform, but in practise the underlying technology may hinder portability. For modelling or describing a component-based system, the Construction and Composition Language (CCL) [9] is used. The CCL is not compliant to any standards. PECT is highly introducible, in principle it should be possible to analyse a part of an existing system using PECT. It should be possible to gradually model larger parts of a system using PECT. A system constructed using PECT can be difficult to understand; mainly because of the mapping from the abstract component model to the concrete component technology. It is likely that systems looking identical at the PECT-level behave differently when realised on different component technologies.

[2] Software Engineering Institute, CMU; http://www.sei.cmu.edu

PECT is an abstract technology that requires an underlying component technology. For instance, how testable and debugable a system is depends on the technical solutions in the underlying run-time system. Resource consumption, computational model, reusability, maintainability, black- or white-box components, static- or dynamic-configuration are also not possible to determine without knowledge of the underlying component technology.

10.3.2 Koala

The Koala component technology [10] is designed and used by Philips[3] for development of software in consumer electronics. Typically, consumer electronics are resource constrained since they use cheap hardware to keep development costs low. Koala is a light weight component technology, tailored for Product Line Architectures [11]. The Koala components can interact with the environment, or other components, through explicit interfaces. The components source code is fully visible for the developers, i.e., there are no binaries or other intermediate formats. There are two types of interfaces in the Koala model, the provides- and the requires- interfaces, with the same meaning as in UML 2.0 [4]. The provides interface specify methods to access the component from the outside, while the required interface defines what is required by the component from its environment. The interfaces are statically connected at design time.

One of the primary advantages with Koala is that it is resource constrained. In fact, low resource consumption was one of the requirements considered when Koala was created. Koala use passive components allocated to active threads during compile-time; they interact through a pipes-and-filters model. Koala uses a construction called thread pumps to decrease the number of processes in the system. Components are stored in libraries, with support for version numbers and compatibility descriptions. Furthermore components can be parameterised to fit different environments.

Koala does not support analysis of run-time properties. Research has presented how properties like memory usage and timing can be predicted in general component-based systems, but the thread pumps used in Koala might cause some problems to apply existing timing analysis theories. Koala has no explicit support for testing and debugging, but they use source code components, and a simple interaction model. Furthermore, Koala is implemented for a specific operating system. A specific compiler is used, which routes all inter-component

[3] Phillips International, Inc; Home Page http://www.phillips.com

and component to operating system interaction through Koala connectors. The modelling language is defined and developed in-house, and it is difficult to see an easy way to gradually introduce the Koala concept.

10.3.3 Rubus Component Model

The Rubus Component Model (Rubus CM) [12] is developed by Arcticus systems.[4] The component technology incorporates tools, e.g., a scheduler and a graphical tool for application design, and it is tailored for resource constrained systems with real-time requirements. The Rubus Operating System (Rubus OS) [13] has one time-triggered part (used for time-critical hard real-time activities) and one event-triggered part (used for less time-critical soft real-time activities). However, the Rubus CM is only supported by the time-triggered part.

The Rubus CM runs on top of the Rubus OS, and the component model requires the Rubus configuration compiler. There is support for different hardware platforms, but regarding to the requirement of portability (Section 10.2.1), this is not enough since the Rubus CM is too tightly coupled to the Rubus OS. The Rubus OS is very small, and all component and port configuration is resolved off-line by the Rubus configuration compiler.

Non-functional properties can be analysed during design-time since the component technology is statically configured, but timing analysis on component and node level (i.e., schedulability analysis) is the only analysable property implemented in the Rubus tools. Testability is facilitated by static scheduling (which gives predictable execution patterns). Testing the functional behaviour is simplified by the Rubus Windows simulator, enabling execution on a regular PC.

Applications are described in the Rubus Design Language, which is a non-standard modelling language. The fundamental building blocks are passive. The interaction model is the desired pipes-and-filters (Section 10.2.1). The graphical representation of a system is quite intuitive, and the Rubus CM itself is also easy to understand. Complexities such as schedule generation and synchronisation are hidden in tools.

The components are source code and open for inspection. However, there is no support for debugging the application on the component level. The components are very simple, and they can be parameterised to improve the possibility to change the component behaviour without changing the component source code. This enhances the possibilities to reuse the components.

[4] Arcticus Systems; Home Page http://www.arcticus.se

Smaller pieces of legacy code can, after minor modifications, be encapsulated in Rubus components. Larger systems of legacy code can be executed as background service (without using the component concept or timing guarantees).

10.3.4 PBO

Port Based Objects (PBO) [14] combines object oriented design, with port automaton theory. PBO was developed as a part of the Chimera Operating System (Chimera OS) project [15], at the Advanced Manipulators Laboratory at Carnegie Mellon University.[5] Together with Chimera, PBO forms a framework aimed for development of sensor-based control systems, with specialisation in reconfigurable robotics applications. One important goal of the work was to hide real-time programming and analysis details. Another explicit design goal for a system based on PBO was to minimise communication and synchronisation, thus facilitating reuse.

PBO implements analysis for timeliness and facilitates behavioural models to ensure predictable communication and behaviour. However, there are few additional analysis properties in the model. The communication and computation model is based on the pipes-and-filters model, to support distribution in multiprocessor systems the connections are implemented as global variables. Easy testing and debugging is not explicitly addressed. However, the technology relies on source code components and therefore testing on a source code level is achievable. The PBOs are modular and loosely coupled to each other, which admits easy unit testing. A single PBO-component is tightly coupled to the Chimera OS, and is an independent concurrent process.

Since the components are coupled to the Chimera OS, it can not be easily introduced in any legacy system. The Chimera OS is a large and dynamically configurable operating system supporting dynamic binding, it is not resource constrained.

PBO is a simple and intuitive model that is highly understandable, both at system level and within the components themselves. The low coupling between the components makes it easy to modify or replace a single object. PBO is built with active and independent objects that are connected with the pipes-and-filters model. Due to the low coupling between components through simple communication and synchronisation the objects can be considered to be highly reusable. The maintainability is also affected in a good way due to the

[5] Carnegie Mellon University; Home Page http://www.cmu.edu

loose coupling between the components; it is easy to modify or replace a single component.

10.3.5 PECOS

PECOS[6] (PErvasive COmponent Systems) [16, 17] is a collaborative project between ABB Corporate Research Centre[7] and academia. The goal for the PECOS project was to enable component-based technology with appropriate tools to specify, compose, validate and compile software for embedded systems. The component technology is designed especially for field devices, i.e., reactive embedded systems that gathers and analyse data via sensors and react by controlling actuators, valves, motors etc. Furthermore, PECOS is analysable, since much focus has been put on non-functional properties such as memory consumption and timeliness.

Non-functional properties like memory consumption and worst-case execution-times are associated with the components. These are used by different PECOS tools, such as the composition rule checker and the schedule generating and verification tool. The schedule is generated using the information from the components and information from the composition. The schedule can be constructed off-line, i.e., a static pre-calculated schedule, or dynamically during run-time.

PECOS has an execution model that describes the behaviour of a field device. The execution model deals with synchronisation and timing related issues, and it uses Petri-Nets [18] to model concurrent activities like component compositions, scheduling of components, and synchronisation of shared ports [19]. Debugging can be performed using COTS debugging and monitoring tools. However, the component technology does not support debugging on component level as described in Section 10.2.1.

The PECOS component technology uses a layered software architecture, which enhances portability (Section 10.2.1). There is a Run-Time Environment (RTE) that takes care of the communication between the application specific parts and the real-time operating system. The PECOS component technology uses a modelling language that is easy to understand, however no standard language is used. The components communicate using a data-flow-oriented interaction, it is a pipes-and-filters concept, but the component technology uses a shared memory, contained in a blackboard-like structure.

[6] PECOS Project, Home Page: http://www.pecos-project.org/
[7] ABB Corporate Research Centre in Ladenburg, Home Page: http://www.abb.com/

Since the software infrastructure does not depend on any specific hardware or operating system, the requirement of introducability (Section 10.2.2) is to some extent fulfilled. There are two types of components, leaf components (black-box components) and composite components. These components can be passive, active, and event triggered. The requirement of openness is not considered fulfilled, due to the fact that PECOS uses black-box components. In later releases, the PECOS project is considering to use a more open component model [20]. The devices are statically configured.

10.3.6 CORBA Based Technologies

The Common Object Request Broker Architecture (CORBA) is a middleware architecture that defines communication between nodes. CORBA provides a communication standard that can be used to write platform independent applications. The standard is developed by the Object Management Group[8] (OMG). There are different versions of CORBA available, e.g., MinimumCORBA [21] for resource constrains systems, and RT-CORBA [22] for time-critical systems.

RT-CORBA is a set of extensions tailored to equip Object Request Brokers (ORBs) to be used for real-time systems. RT-CORBA supports explicit thread pools and queuing control, and controls the use of processor, memory and network resources. Since RT-CORBA adds complexity to the standard CORBA, it is not considered very useful for resource-constrained systems. MinimumCORBA defines a subset of the CORBA functionality that is more suitable for resource-constrained systems, where some of the dynamics is reduced.

OMG has defined a CORBA Component Model (CCM) [23], which extends the CORBA object model by defining features and services that enables application developers to implement, mange, configure and deploy components. In addition the CCM allows better software reuse for server-applications and provides a greater flexibility for dynamic configuration of CORBA applications.

CORBA is a middleware architecture that defines communication between nodes, independent of computer architecture, operating system or programming language. Because of the platform and language independence CORBA becomes highly portable. To support the platform and language independence, CORBA implements an Object Request Broker (ORB) that during run-time acts as a virtual bus over which objects transparently interact with other objects located locally or remote. The ORB is responsible for finding a requested

[8] Object Management Group. CORBA Home Page. http://www.omg.org/corba/

objects implementation, make the method calls and carry the response back to the requester, all in a transparent way. Since CORBA run on virtually any platform, legacy code can exist together with the CORBA technology. This makes CORBA highly introducible.

While CORBA is portable, and powerful, it is very run-time demanding, since bindings are performed during run-time. Because of the run-time decisions, CORBA is not very deterministic and not analysable with respect to timing and memory consumption. There is no explicit modelling language for CORBA. CORBA uses a client server model for communication, where each object is active. There are no non-functional properties or any specification of interface behaviour. All these things together make reuse harder. The maintainability is also suffering from the lack of clearly specified interfaces.

10.4 Summary of Evaluation

In this section we assign numerical grades to each of the component technologies described in Section 10.3, grading how well they fulfil each of the requirements of Section 10.2. The grades are based on the discussion summarised in Section 10.3. We use a simple 3 level grade, where 0 means that the requirement is not addressed by the technology and is hence not fulfilled, 1 means that the requirement is addressed by the technology and/or that is partially fulfilled, and 2 means that the requirement is addressed and is satisfactory fulfilled. For PECT, which is not a complete technology, several requirements depended on the underlying technology. For these requirements we do not assign a grade (indicated with NA, Not Applicable, in Figure 10.1). For the CORBA-based technologies we have listed the best grade applicable to any of the CORBA flavours mentioned in Section 10.3.6.

For each requirement we have also calculated an average grade. This grade should be taken with a grain of salt, and is only interesting if it is extremely high or extremely low. In the case that the average grade for a requirement is extremely low, it could either indicate that the requirement is very difficult to satisfy, or that component-technology designers have paid it very little attention.

In the table we see that only two requirements have average grades below 1.0. The requirement "Component Modelling" has the grade 0 (!), and "Testing and debugging" has 1.0. We also note that no requirements have a very high grade (above 1.5). This indicate that none of the requirement we have listed are general (or important) enough to have been considered by all component-

technology designers. However, if ignoring CORBA (which is not designed for embedded systems) and PECT (which is not a complete component technology) we see that there are a handful of our requirements that are addressed and at least partially fulfilled by all technologies.

We have also calculated an average grade for each component technology. Again, the average cannot be directly used to rank technologies amongst each other. However, the two technologies PBO and CORBA stand out as having significantly lower average values than the other technologies. They are also distinguished by having many 0's and few 2's in their grades, indicating that they are not very attractive choices. Among the complete technologies with an average grade above 1.0 we notice Rubus and PECOS as being the most complete technologies (with respect to this set of requirements) since they have the fewest 0's. Also, Koala and PECOS can be recognised as the technologies with the broadest range of good support for our requirements, since they have the most number of 2's.

However, we also notice that there is no technology that fulfils (not even partially) all requirements, and that no single technology stands out as being the preferred choice.

	Analysable	Testable and debugable	Portable	Resource Constrained	Component Modelling	Computational Model	Introducible	Reusable	Maintainable	Understandable	Source Code Components	Static Configuration	Average	Number of 2's	Number of 0's
PECT	2	NA	2	NA	0	NA	2	NA	NA	0	NA	NA	1.2	3	2
Koala	0	1	1	2	0	2	0	2	2	2	2	2	1.3	7	3
Rubus Component Model	1	1	0	2	0	2	1	1	1	2	2	2	1.3	5	2
PBO	2	1	0	0	0	1	1	1	1	2	2	0	0.9	3	4
PECOS	2	1	2	2	0	2	1	2	1	2	0	2	1.4	7	2
CORBA Based Technologies	0	1	2	0	0	0	2	0	0	1	0	0	0.5	2	8
Average	1.2	1.0	1.2	1.2	0.0	1.4	1.4	1.2	1.0	1.5	1.2	1.2	1.1	4.3	3.5

Figure 10.1: Grading of component technologies with respect to the requirements

10.5 Conclusion

In this paper we have compared some existing component technologies for embedded systems with respect to industrial requirements. The requirements have been collected from industrial actors within the business segment of heavy vehicles. The software systems developed in this segment can be characterised as resource constrained, safety critical, embedded, distributed, real-time, control systems. Our findings should be applicable to software developers whose systems have similar characteristics.

We have noticed that, for a component technology to be fully accepted by industry, the whole systems development context needs to be considered. It is not only the technical properties, such as modelling, computation model, and openness, that needs to be addressed, but also development requirements like maintainability, reusability, and to which extent it is possible to gradually introduce the technology. It is important to keep in mind that a component technology alone cannot be expected to solve all these issues; however a technology can have more or less support for handing the issues.

The result of the investigation is that there is no component technology available that fulfil all the requirements. Further, no single component technology stands out as being the obvious best match for the requirements. Each technology has its own pros and cons. It is interesting to see that most requirements are fulfilled by one or more techniques, which implies that good solutions to these requirements exist.

The question, however, is whether it is possible to combine solutions from different technologies in order to achieve a technology that fulfils all listed requirements? Our next step is to assess to what extent existing technologies can be adapted in order to fulfil the requirements, or whether selected parts of existing technologies can be reused if a new component technology needs to be developed. Examples of parts that could be reused are file and message formats, interface description languages, or middleware specifications/implementations. Further, for a new/modified technology to be accepted it is likely that it have to be compliant to one (or even more than one) existing technology. Hence, we will select one of the technologies and try to make as small changes as possible to that technology.

Bibliography

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

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

[3] B. Selic and J. Rumbaugh. Using UML for modelling complex real-time systems, 1998. Rational Software Corporation.

[4] Object Management Group. UML 2.0 Superstructure Specification, The OMG Final Adopted Specification, 2003. http://www.omg.com/uml/.

[5] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall; 1 edition, 1996. ISBN 0-131-82957-2.

[6] D. Garlan, R. Allen, and J. Ockerbloom. Architectural mismatch or why it's hard to build systems out of existing parts. In *Proceedings of the Seventeenth International Conference on Software Engineering*, April 1995.

[7] Microsoft Component Technologies. Com/dcom/.net. http://www.microsoft.com.

[8] R. Monson-Haefel. *Enterprise JavaBeans, Third Edition*. O'Reilly & Assiciates, Inc., 2001. ISBN: 0-596-00226-2.

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

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

[11] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2001. ISBN 0-201-70332-7.

[12] K.L. Lundbäck and J. Lundbäck and M. Lindberg. Component-Based Development of Dependable Real-Time Applications. Arcticus Systems: http://www.arcticus.se (Last Accessed: 2005-01-18).

[13] K.L. Lundbäck. Rubus OS Reference Manual – General Concepts. Arcticus Systems: http://www.arcticus.se (Last Accessed: 2005-01-18).

[14] D.B. Stewart, R.A. Volpe, and P.K. Khosla. Design of Dynamically Reconfigurable Real-Time Software Using Port-Based Objects. *IEEE Transactions on Software Engineering*, pages pages 759 – 776, December 1997.

[15] P.K. Khosla et al. The Chimera II Real-Time Operating System for Advanced Sensor-Based Control Applications. *IEEE Transactions on Systems*, 1992. Man and Cybernetics.

[16] M. Winter, T. Genssler, et al. Components for Embedded Software – The PECOS Apporach. In *The Second International Workshop on Composition Languages, in conjunction with the 16th ECOOP*, June 2002.

[17] T. Genssler, A. Christoph, B. Schulz, M. Winter, C. M. Stich, C. Zeidler, P. Müller, A. Stelter, O. Nierstrasz, S. Ducasse, G. Arévalo, R. Wuyts, P. Liang, B. Schönhage, and R. van den Born. Pecos in a nutshell. Technical report, Technical report, The PECOS Consortium, 2002.

[18] M. Sgroi. Quasi-Static Scheduling of Embedded Software Using Free-Choice Petri Nets. Technical report, University of California at Berkely, May 1998.

[19] O. Nierstrass, G. Arevalo, S. Ducasse, , R. Wuyts, A. Black, P. Müller, C. Zeidler, T. Genssler, and R. van den Born. A Component Model for Field Devices. In *Proceedings of the First International IFIP/ACM Working Conference on Component Deployment*, June 2002.

[20] R. Wuyts and S. Ducasse. Non-functional requirements in a component model for embedded systems. In *International Workshop on Specification and Verification of Component-Based Systems*, 2001. OPPSLA.

[21] Object Management Group. MinimumCORBA 1.0. http://www.omg.org/technology/documents/formal/minimum_CORBA.htm (Last Accessed: 2005-01-17).

[22] D.C. Schmidt, D.L. Levine, and S. Mungee. The Design of the tao real-time object request broker. *Computer Communications Journal*, Summer 1997.

[23] OMG. Corba component model 3.0. Technical report, OMG Documatation (ccm/02-04-01), April 2002.