Mälardalen University Press Licentiate Theses
No. 146

# MODELING AND TIMING ANALYSIS OF INDUSTRIAL COMPONENT-BASED DISTRIBUTED REAL-TIME EMBEDDED SYSTEMS

**Saad Mubeen**

**2012**


School of Innovation, Design and Engineering

ISBN 978-91-7485-055-0
ISSN 1651-9256
Printed by Mälardalen University, Västerås, Sweden

# Abstract

The model- and component-based development approach has emerged as an attractive option for the development of Distributed Real-time Embedded (DRE) systems. Within this context we target issues related to modeling of legacy communication, extraction of end-to-end timing models and support for holistic response-time analysis of industrial DRE control systems.

We introduce a new approach for modeling legacy network communication in component-based DRE systems. By introducing special-purpose components to encapsulate and abstract the communication protocols in DRE systems, we allow the use of legacy nodes and legacy protocols in a component- and model-based software engineering environment. The proposed approach also supports the state-of-the-practice development of component-based DRE systems. Because an end-to-end timing model should be available to perform the holistic response-time analysis, we present a method to extract the end-to-end timing models from component-based DRE systems.

The Controller Area Network (CAN) is one of the widely used real-time networks in DRE systems especially in automotive domain. We identify that the existing analysis of CAN does not support common message transmission patterns which are implemented by some high-level protocols used in the industry. Consequently, we extend the existing analysis to facilitate the worst-case response-time calculation of these transmission patterns. The extended analysis is generally applicable to any high-level protocol for CAN that uses periodic, sporadic, and both periodic and sporadic transmission of messages.

In order to show the applicability of our modeling techniques and extended analysis, we provide a proof of concept by extending the existing industrial component model (Rubus Component Model), implementing the holistic response-time analysis along with the extended analysis of CAN in the industrial tool suite (Rubus-ICE), and conducting an automotive-application case study.

To my mother

# Acknowledgements

First of all, I would like to express my deepest gratitude to my supervisors Professor Mikael Sjödin and Dr. Jukka Mäki-Turja. The work presented in this thesis would not have been possible without their expert guidance, persistent help and tremendous encouragement. I am grateful to them for providing valuable and useful suggestions for improvement of this thesis. I had a great opportunity of learning so many new things from them during our meetings and discussions.

Many thanks to the people from industry who were involved in the work presented in this thesis. Thank you Kurt-Lennart Lundbäck, John Lundbäck, Staffan Sandberg and Jimmy Westerlund.

I would like to thank Dr. Jan Carlson for co-authoring a paper and providing me useful feedback on my thesis proposal. I also thank Farhang Nemati for providing me useful tips on the structure of my thesis.

I attended several courses during my Licentiate studies. I thank Hans Hansson, Thomas Nolte, Emma Nehrenheim, Mikael Sjödin, Jukka Mäki-Turja, Ivica Crnkovic, Jan Torin, Sasikumar Punnekkat, and Kristina Lundqvist for their guidance during my studies. I want to also thank other faculty members Paul Pettersson, Jan Gustafsson, Björn Lisper, Mats Björkman, Jan Carlson, Damir Isovic, Dag Nyström, Cristina Seceleanu, Gordana Dodig-Crnkovic, Mikael Ekström, Andreas Ermedahl. You all have been a source of inspiration for me.

I would also like to thank my friends and colleagues at the department for all the fun we had during my studies, conference trips, coffee breaks and parties. I wish to thank Abhilash, Adam, Adnan, Aida, Amine,Ana, Andreas G., Andreas H., Andreas J., Aneta, Antonio, Barbara, Batu, Bob (Stefan), Danial, Eduard, Etienne, Farhang, Federico, Frank, Giacomo, Hang, Huseyin, Jagadish, Johan, Josip, Juraj, Jörgen, Lars, Leo, Luis (Yue), Luka, Mehrdad, Mikael Å, Mobyen, Moris, Nikola, Nima, Ning, Radu, Rafia, Raluca, Sara D.,

Severine, Shahina, Stefan B., Svetlana, Thomas L., Tibi, and others for all the fun and memories.

I also thank all the administrative staff, in particular Gunnar Widforss, Malin Rosqvist, Åsa Lundkvist, Carola Ryttersson, Sussane Fronnå for making many things easier.

Last but not least, I would like to thank my family. I thank my parents for their endless love, support and encouragement throughout my life. I am thankful to my wife for her care, support and cooperation.

This work has been supported by the Swedish Knowledge Foundation (KKS) within the project EEMDEF and the Swedish Foundation for Strategic Research (SSF) with the centre PROGRESS. I would like to thank the industrial partners Arcticus Systems and BAE Systems Hägglunds.

Saad Mubeen
Västerås, January, 2012

# List of Publications

## Papers Included in the Licentiate Thesis[1]

**Paper A** *Analyzable Modeling of Legacy Communication in Component Based Distributed Embedded Systems*. Saad Mubeen, Jukka Mäki-Turja, Mikael Sjödin and Jan Carlson. In proceedings of the $37^{th}$ Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pages 229-238, Oulu, Finland, September, 2011.

**Paper B** *Extraction of End-to-end Timing Model from Component- Based Distributed Real-Time Embedded Systems*. Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin. In proceedings of the International Workshop on Time Analysis and Model-Based Design, from Functional Models to Distributed Deployments (TiMoBD) located at Embedded Systems Week, Taipei, Taiwan, October, 2011.

**Paper C** *Extending Schedulability Analysis of Controller Area Network (CAN) for Mixed (Periodic/Sporadic) Messages*. Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin. In proceedings of the $16^{th}$ IEEE Conference on Emerging Technologies and Factory Automation (ETFA), pages 1-10, Toulouse, France, September, 2011.

**Paper D** *Support for Holistic Response-time Analysis in an Industrial Tool Suite: Implementation Issues, Experiences and a Case Study*. Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin. Accepted for publication in proceedings of the $19^{th}$ IEEE Conference on Engineering of Computer Based Systems (ECBS), Novi Sad, Serbia, April, 2012.

[1] The included articles have been reformatted to comply with the licentiate layout

# Additional Papers, Not Included in the Licentiate Thesis

## Journals

- *Introducing Components for Modeling Real-Time Network Communication in the Rubus Component Model*. Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin. Accepted for publication in the Information Journal, International Information Institute, March, 2012.

- *Tracing Event Chains for Holistic Response-Time Analysis of Component Based Distributed Real-Time Systems*. Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin. In the ACM SIGBED Review, vol. 8, issue 3, pages 48-51, ACM, September, 2011.

## Conferences

- *Response-Time Analysis of Mixed Messages in Controller Area Network with Priority- and FIFO-Queued Nodes*. Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin. In submission.

- *Towards Modeling and Holistic Timing Analysis of Industrial Component Based DRE Systems*. Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin. Accepted for publication in proceedings of the $19^{th}$ IEEE Conference on Engineering of Computer Based Systems (ECBS), Novi Sad, Serbia, April, 2012.

- *Implementation of Holistic Response-Time Analysis in Rubus-ICE: Preliminary Findings, Issues and Experiences*. Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin. In proceedings of the $32^{nd}$ IEEE Real-Time Systems Symposium (RTSS), WIP, pages 9-12, Vienna, Austria, December, 2011.

- *Extending Response-Time Analysis of Controller Area Network (CAN) with FIFO Queues for Mixed Messages*. Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin. In proceedings of the $16^{th}$ IEEE Conference on Emerging Technologies and Factory Automation (ETFA), pages 1-4, Toulouse, France, September, 2011.

- *Exploring Options for Modeling of Real-Time Network Communication in an Industrial Component Model for Distributed Embedded Systems*. Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin. In the Lecture Notes in Electrical Engineering (LNEE), Vol. 102, Springer, pages 441-458, August, 2011.

- *Designing Efficient Source Routing for Mesh Topology Network on Chip Platforms*. Saad Mubeen and Shashi Kumar. In proceedings of the $13^{th}$ Euromicro Conference on Digital System Design, Architectures, Methods and Tools (DSD), pages 181-188, Lille, France, September, 2010.

- *High Precision Response Time Analysis of Tasks with Precedence Chains*. Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin. In proceedings of the $22^{nd}$ Euromicro Conference on Real-Time Systems (ECRTS), WIP, pages 21-24, Brussels, Belgium, July, 2010.

## Workshop

- *Modeling of Legacy Communication in Distributed Embedded Systems*. Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin. In proceedings of the $2^{nd}$ Workshop on Model Based Engineering for Embedded Systems Design (M-BED), located at Design, Automation & Test in Europe (DATE) Conference, pages 1-6, Grenoble, France, March, 2011.

## MRTC reports

- *Implementation of Holistic Response-time Analysis in Rubus-ICE*. Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin. Technical Report ISSN 1404-3041 ISRN MDH-MRTC-258/2012-1-SE, Mälardalen University, Sweden, January, 2012.

- *Response-Time Analysis of Mixed-Type Controller Area Network (CAN) Messages*. Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin. Technical Report ISSN 1404-3041 ISRN MDH-MRTC-259/2012-1-SE, Mälardalen University, Sweden, January, 2012.

# Contents

# I

# Thesis

# Chapter 1

# Introduction

In this thesis we introduce a new approach for modeling legacy network communication in component-based Distributed Real-time Embedded (DRE) systems. By introducing special-purpose components to encapsulate and abstract the communication protocols in DRE systems, we allow the use of legacy nodes and legacy protocols in a component- and model-based software engineering environment. The proposed approach also supports the state-of-the-practice development of component-based DRE systems. Because an end-to-end timing model should be available to perform the holistic response-time analysis, we also provide a method to extract such models from component-based DRE systems.

## 1.1 Background

An embedded system is a microprocessor-based system that is designed to perform a dedicated functionality by means of hardware and software [1]. Often, embedded systems interact with their environment through sensors and actuators. They mostly remain hidden in their applications, for example, an embedded system in a vending machine, because they are embedded inside the larger system which they control or which they are part of. They are found in almost all electronic items ranging from simple consumer products such as microwave oven and coffee machine to highly sophisticated systems such as industrial process controllers and smart phones. Their applications span over many domains such as automotive, aerospace, consumer electronics, biomedi-

cal, military, business, industrial control, and many more.

It is estimated that about 10 billion processors are manufactured every year. Out of which, approximately 99% are embedded processors while only 1% find their way to the general-purpose computers such as PCs and laptops [1, 2]. Not only the number of embedded processors has increased in the past few years, but also the software which runs on them. The embedded software has drastically increased in size and complexity. In automotive domain, for example, a modern premium car contains nearly 100 million lines of code that run on about 70 to 100 embedded processors [3]. Another example of the complexity and large size of embedded software can be seen in the software for radio and navigation system in a modern premium car such as Mercedes Benz S-Class that alone contains 20 million lines of code [3]. Because of this trend of continuously increasing size and complexity of embedded software, the development of embedded systems has become very complex.

Often, an embedded system needs to interact with its environment in a timely manner, i.e., the embedded system is a real-time system. For such a system, the desired and correct output is one which is logically correct as well as delivered within a specified time (e.g., a deadline). One way to classify a real-time system is as being either soft or hard. In soft real-time systems, infrequent deadline misses can be tolerated. For example, electronic window control system in a car is a soft real-time system. On the other hand, missing a deadline in a hard real-time system can result in the system failure. In hard real-time systems, a logically correct but late response is considered as bad as logically incorrect response. The electronic engine control system in a car is an example of a hard real-time system. Many hard real-time systems are also safety critical which means that the system failure can result in catastrophic consequences such as endangering human life or the environment. For example, airbag control system in a car is a safety-critical hard real-time system.

In order to capture, e.g., requirements early during the development, handle the complexity of embedded software, lower the development cost, reduce the time-to-market and time-to-test, allow reusability, and support modeling at higher level of abstraction, the research community proposed model- and component-based development of embedded systems by employing the principles of Model-Based software Engineering (MBE) and Component-Based Software Engineering (CBSE) [4, 5]. MBE provides the means to use models throughout the process of system development. It uses models to describe functions, structures and other design artifacts. Whereas, CBSE facilitates the development of large software systems by integration of software components. CBSE raises the level of abstraction for software development and aims to

reuse software components and their architectures. There is a great interest for bringing these development techniques in the embedded systems industry [5, 6].

In DRE systems, the functionality is distributed over many nodes (processors). The nodes in a DRE system are connected to one or more networks. The software development of DRE systems is much more complex compared to uniprocessor embedded real-time systems because of various reasons including the distribution of functionality and real-time requirements on network communications. The example of a modern premium car, that we discussed above, provides a good example of an application of DRE systems. The size of embedded software in a modern premium car may reach up to 1 GB which may be realized by more than 2000 software functions distributed over 70 to 100 Electronic Control Units (ECUs) that may be connected by more than five different buses (or networks) [7].

When MBE and CBSE are used for the development of DRE systems, modeling of communication infrastructure arises as a challenge. In the industry, DRE systems are built often using legacy (sub) systems (i.e., previously developed) which use predefined rules for communication. Furthermore, DRE systems are often expected to use legacy network protocols for real-time communication. A component technology for the development of DRE systems should abstract the application software from the communication infrastructure. Moreover, the component technology should support the modeling and analysis of legacy communications and legacy systems.

The safety-critical nature of many DRE systems requires evidence that the actions by the system will be provided in a timely manner, i.e., each action will be taken at a time that is appropriate to the environment of the system. Therefore, it is important to make accurate predictions of the timing behavior of such systems. In order to provide evidence that each action in the system will meet its deadline, *a priori* analysis techniques such as schedulability analysis have been developed by the research community. The Holistic Response-Time Analysis (HRTA) [8] is a schedulability analysis technique which calculates upper bounds on the response times of event chains that are distributed over more than one node in a DRE system. The end-to-end timing model of a DRE system should be available to perform HRTA. Ideally, a component technology for the development of DRE systems should support automatic extraction of such timing model.

There are a number of real-time network protocols used in DRE systems. Among them, Controller Area Network (CAN) [9] is one of the most frequently used especially in automotive domain. It has been standardized by the Inter-

national Organization for Standardization as ISO 11898-1 [10]. According to CAN in Automation (CiA) [11], the number of CAN enabled controllers sold in 2011 are estimated to be 850 million. In total, more than two billion CAN controllers have been sold until today. Out of this huge number, approximately 80% CAN controllers have been used in automotive domain. CAN is a multi-master, event-triggered, serial communication bus protocol supporting bus speeds of up to 1 mega bits per second. In this thesis, we will focus only on CAN and some of its high-level protocols which are developed for various industrial applications. These include CAN Application Layer (CAL) [12], CANopen [13], Hägglunds Controller Area Network (HCAN) [14], CAN for Military Land Systems domain (MilCAN) [15], etc.

## 1.2 Problem Statement and Research Questions

The model- and component-based development has emerged as an attractive option for the development of software for DRE systems. The majority of existing model- and component-based development approaches allow for structural and functional modeling. They do not support execution modeling which is concerned with the modeling of run-time properties and/or requirements (e.g., end-to-end deadlines, jitter, etc.) of software functions. The modeling of DRE systems should extend down to the execution level to allow precise control of resource utilization and that timing requirements are not violated when the system is executed. However, providing such modeling support for DRE systems is very challenging because the functionality in DRE systems can be realized with more than one execution model, e.g., separate execution models for the nodes and networks. Today, one of the main focus points during the development of DRE systems in the industry is to model and express timing related information and perform timing analysis [16].

One way to deal with these challenges is to use a component technology that allows model- and component-based development of DRE systems with the support for modeling, analyzing, predicting and modifying the execution behavior. Such a component technology should complement structural and functional modeling with the modeling of execution requirements at an abstraction level close to the functional specification while abstracting the implementation details. The component technology should allow the expression of timing related information during the development. Moreover, it should facilitate the identification of timing errors early during the development by easily rendering the modeled DRE applications for end-to-end timing analysis.

However, building such a component technology to support the state-of-the-practice development of DRE systems raises many challenges. One of the main reasons behind these challenges is that the development process of DRE systems in academia and industry may be very different from each other. In academia, the development process often starts with discussions about models and functions. The models are assumed to be platform independent. Further, it is assumed that the models and functions will be deployed on specific platforms at a later stage. However, this way of development for DRE systems is often not practiced in the industry, especially in automotive or vehicle domain. The traditional process for the development of DRE systems in the industry starts with designing the bus (or network) communication. The infrastructure for the DRE system to be developed is already known. In the early stage of industrial development process of DRE systems, usually the focus is on finding the answers to the questions as follows. How many busses will be there in the system? Which nodes will be connected to which bus? How many messages will be there in the system? Which messages will be sent by each node? After finding the answers to these questions, the focus is shifted towards the development of functions. Thus, a communication-oriented development process is used for DRE systems and constitutes the state of the practice.

In order to provide a model- and component-based approach to support the state-of-the-practice development of DRE systems, we will target the challenges concerned with the modeling of real-time network communication and support for holistic timing analysis. One such challenge is to support the modeling of legacy network communication and allow the use of legacy nodes in component-based DRE systems. In order to ensure that the DRE system will behave in a timely manner during its execution, we need to analyze tasks, messages and event chains in distributed transactions and predict the end-to-end delays. The component technology for the industrial development of DRE systems should support state-of-the-art real-time analysis such as Holistic Response-Time Analysis (HRTA). The supported HRTA should be able to incorporate the analysis of common message transmission patterns that are implemented by the real-time network protocols used in the industry. In order to perform HRTA, the end-to-end timing model of DRE systems should be available. The extraction of end-to-end timing model from component-based DRE systems is another challenge that we will target.

The research problem addressed in this thesis can be formulated as follows.

> *Investigate how to provide a model- and component-based approach for communications-oriented development of DRE systems with a support for legacy communication protocols, legacy nodes and holistic response-time analysis.*

We further refine this problem to formulate two questions that we will investigate in this thesis.

1. How to model legacy network communication and allow the use of legacy nodes for the state-of-the-practice development processes for component-based DRE systems?
2. How to extract end-to-end timing models from component-based DRE systems that are built using the state-of-the-practice development processes?

## 1.3 Thesis Outline

The thesis is organized into two parts:
**Part I** includes first three chapters. In Chapter 1 we provided an introduction to the thesis and formulated the research problem. In Chapter 2 we discuss the contributions in the thesis. Chapter 3 presents the conclusion and suggestions for the future work.
**Part II** presents the technical contributions of the thesis in the form of four papers which are organized in Chapters 4-7.

# Chapter 2

# Technical Contributions

This thesis presents the development and implementation of new modeling and timing analysis techniques which can be used for the state-of-the-practice development of component-based DRE systems. The contributions in this thesis are organized in four parts. In the first part, we introduce a new technique for modeling legacy network communication in DRE systems. The detailed contribution in this part is discussed in Paper A (Chapter 4). In the second part, we present a method to extract the end-to-end timing models from component-based DRE systems. The detailed contribution in this part is discussed in Paper B (Chapter 5). In the third part, we identify a need for the extension of existing response-time analysis of CAN, and accordingly, we present the extended analysis. The detailed contribution in this part is discussed in Paper C (Chapter 6). Finally, in the fourth part, we provide a proof-of-concept implementation of the techniques developed in previous three parts. The detailed contribution in the fourth part is discussed in Paper D (Chapter 7). In this chapter we provide a summary of these contributions.

**Personal Contribution.** The research work presented in these contributions was done in collaboration with my supervisors Prof. Mikael Sjödin and Dr. Jukka Mäki-Turja along with Dr. Jan Carlson (only Paper A). I am the main contributor and first author of all the papers.

## 2.1 Modeling of Legacy Network Communication in Component-based DRE Systems

This contribution addresses first research question. We introduce a new approach for modeling real-time network and legacy communication in component-based DRE systems. In order to show usability of our modeling approach, we implement it by extending the existing industrial component model, i.e., Rubus Component Model (RCM) [17]. By introducing special-purpose components to encapsulate and abstract the communication protocols in DRE systems, we allow the use of legacy nodes and legacy protocols in a component- and model-based software engineering environment. With the addition of these components, RCM will be able to not only model real-time network communication, but also support state-of-the-practice development of component-based DRE systems. The proposed extension also allows model- and component-based development of new nodes that are deployed in legacy systems that use predefined communication rules. These extensions also enable adaptation of a node when communication rules change (e.g., due to re-deployment in a new system or due to upgrades in the communication system) without affecting its internal component design. The special-purpose components can be automatically generated from the information about legacy communication or from early design decisions about network communication. Although RCM was selected for the proof-of-concept implementation, the proposed extensions should be generally applicable for the extension of several component models for the development of DRE systems that use the pipe-and-filter style for component interconnection such as ProCom [18] and COMDES-II [19].

## 2.2 Extraction of End-to-end Timing Models

This contribution addresses second research question. HRTA is an important activity during the development of DRE systems. In order to perform HRTA of component-based DRE systems, the end-to-end timing models should be extracted from them. The extraction of such models can be challenging because the design and analysis models are usually built using different meta-models. We present a method to extract the end-to-end timing models from component-based DRE systems to facilitate HRTA. This method is built upon the modeling approach that we discussed in the first contribution (Paper A). We discuss and solve the issues concerning the model extraction such as extraction of unambiguous timing and tracing information from all nodes and networks in the

system and tracing of event chains in distributed transactions. The extraction method for end-to-end timing models and the solutions of encountered problems may be applied to several component models that use a pipe-and-filter style for component interconnection. The end-to-end timing model that we considered is also general as it incorporates the analysis of several real-time network protocols used in the automotive domain. To show the applicability of our approach, we demonstrate the implementation of end-to-end timing model extraction in the analysis framework of the existing industrial tool suite Rubus-ICE [20].

## 2.3 Extension of the Existing Analysis for Controller Area Network

To analyze communications in DRE systems, it is important to find out whether the existing analysis is sufficient or extensions are required to meet the industrial needs. In this work, we focus only on CAN and some of its high-level protocols. While answering the two research question (discussed in Chapter 1), we identified that the existing response-time analysis of CAN does not support the analysis of common message transmission patterns which are implemented by some high-level protocols used in the industry. The existing analysis calculates the response times of CAN messages that are queued for transmission periodically or sporadically. However, there are a few high-level protocols for CAN such as CANopen and HCAN that support the transmission of mixed messages as well. A mixed message can be queued for transmission both periodically and sporadically. In other words, a mixed message is simultaneously time and event triggered. Thus, it may not exhibit a periodic activation pattern. In order to support the development of DRE systems employing high-level protocols for CAN, there is a need to extend the existing analysis. We extend the existing response-time analysis of CAN to support mixed messages. The extended analysis is generally applicable to any high-level protocol for CAN that uses periodic, sporadic, and both periodic and sporadic transmission of messages.

## 2.4 Proof-of-Concept Implementation

In this contribution we validate our solutions to the research questions. In order to transfer the new modeling techniques and extended analysis, discussed in the previous three contributions, to the industry we need to validate them first.

While validating our solutions, we found out that the process of implementing and integrating state-of-the-art real-time analysis with an existing industrial tool suite offers many challenges. The Implementer has to not only code and implement the analysis in the tool suite, but also deal with several other issues. We present the implementation of HRTA as a plug-in for the existing industrial tool suite Rubus-ICE. As part of HRTA, we implemented the existing as well as the extended analysis discussed in the third contribution. The implemented HRTA is general as it supports the integration of response-time analysis of various networks without a need for changing the holistic algorithm. We discuss and solve encountered issues and highlight gained experiences during the implementation, integration and evaluation of HRTA plug-in. We believe that most of the experiences gained and solutions to the issues encountered in this work maybe applicable when other complex real-time analysis techniques are implemented in any industrial tool suite that supports a plug-in framework (for the integration of new tools) and component-based development of DRE systems. Finally, we provide a proof of concept for all modeling approaches and extended analysis discussed in this thesis by modeling an automotive industrial application (autonomous cruise control system) using extended RCM and analyzing it with HRTA plug-in in Rubus-ICE.

## 2.5 Discussion

We selected RCM and accompanying tool suite Rubus-ICE to provide a proof-of-concept implementation of our new modeling techniques and extended analysis for several reasons. Among them, one reason is the existing support for structural, functional and execution modeling of dependable embedded real-time systems. Further, RCM and Rubus-ICE provide a means for developing predictable and analyzable control functions with a support for modeling real-time properties and requirements, interconnections between the functions in terms of data flow and control flow separately, and generation of run-time framework.

With the proposed extensions, RCM along with Rubus-ICE can be considered a suitable choice for the component-based development of DRE systems in the industry for many reasons. For example, it complements the structural and functional modeling with the execution modeling of DRE systems; it supports communications-oriented development process for DRE systems; it supports the modeling of legacy communication and legacy systems; it can easily model and specify the timing related information; it has a small run-time

footprint (timing and memory overhead); it implements the state-of-the-art research results; and it has a strong support for development and analysis tools.

## 2.6 Impact of Contributions

The new approaches for modeling legacy network communication and extraction of end-to-end timing models may be suitable for other component models, for DRE systems, that use a pipe-and-filter style for component interconnection. The extended analysis supports common message transmission patterns that are implemented by several high-level protocols used in the industry today. Further, the analysis engines support the integration of the analysis of various real-time networks without a need for changing the holistic algorithm. Most of the encountered issues, proposed solutions and gained experiences in this work may provide guidance for the implementation of other complex real-time analysis in any industrial tool suite that supports a plug-in framework (for the integration of new tools) and component-based development of DRE systems.

The new release of RCM and Rubus-ICE (Version 4.0) incorporates the contributions and results presented in this thesis.

# Chapter 3

# Conclusions

## 3.1 Summary and Conclusions

In this thesis we introduced new techniques to provide a model- and component-based support for communications-oriented development of Distributed Real-time Embedded (DRE) control systems.

In order to provide a solution to the first research question, we proposed a new approach for modeling legacy network communication in component-based DRE systems. The proposed approach abstracts the implementation and configuration of communications in DRE systems. It enables the communication capabilities of a node very explicit, but efficiently hides the implementation or protocol details. Moreover, the new approach allows model- and component-based development of new nodes that are deployed in legacy systems that use predefined communication rules. The proposed approach also enables adaptation of a node when communication rules change without affecting its internal component design. As a solution to our second research question, we presented a method to extract end-to-end timing models from component-based DRE systems that are developed using above modeling approach. The purpose of extracting the end-to-end timing models is to support the Holistic Response Time Analysis (HRTA) of DRE systems.

We believe, these techniques may be suitable for several other component models for DRE systems that use a pipe-and-filter style for component interconnection. Moreover, these techniques can be used for any type of "inter-model signaling", where a signal leaves one model (e.g., a node, or a core, or a process) and appears again in some other model.

While we were looking for answers to our research questions, we identified a need for the extension of existing response-time analysis of CAN to support the analysis of common message transmission patterns that are implemented by some high-level protocols used in the industry. Accordingly, we extended the existing analysis which is generally applicable to any high-level protocol or commercial extension of CAN that uses periodic, sporadic, and both periodic and sporadic transmission of messages.

We provided a proof-of-concept implementation of our modeling and analysis approaches by extending the existing industrial component model, i.e., the Rubus Component Model (RCM); implementing the extended HRTA in an industrial tool suite, i.e., Rubus-ICE; and conducting an automotive-application case study. The analysis engines that we provide are able to predict important execution characteristics of the system such as holistic response times without a need for tedious and expansive testing.

We believe, the industrial tools that implement our modeling techniques and extended analysis for the development of DRE control systems may prove helpful for the software development organizations in the automotive domain to decrease the costs for software development, configuration and testing.

## 3.2 Future Work

An interesting future research direction is to investigate and develop patterns that allow transformation between several domain-specific modeling languages in the vehicular domain. The idea is to bridge the semantic gap between functional models (expressed in standard languages as EAST-ADL [21] and/or proprietary languages such as Simulink [22] or Statemate [23]) and execution models (expressed in standard languages like TADL [24] and Autosar [6] and/or proprietary languages like RCM). It would also be interesting and useful to facilitate the exchange of analysis models and tools between RCM and several other component models and tools used for the development of automotive embedded systems.

Another future work could be extending the existing analysis of CAN by combining the analysis of mixed messages in CAN (presented in this thesis) and analysis of CAN with FIFO queues [25]. The extended analysis will be able to compute the worst-case response times of mixed messages in the CAN network where some nodes use FIFO queues while others use priority queues. The preliminary work in this direction is presented in [26]. Another future work in this direction is the extension of CAN analysis for mixed messages

which have multiple sources for periodic and sporadic triggering.

In the future, the HRTA plug-in can be expanded by implementing and integrating the analysis of other network communication protocols (e.g., Flexray, switched ethernet, etc.) within the holistic analysis algorithms discussed in this thesis. Another future work could be providing a support for asynchronous data-flow using the two different semantics of data-age and reaction (described in [27]) in Rubus-ICE.

# Bibliography


[1] Michael Barr and Anthony Massa. *Programming Embedded Systems*. O'Reilly Media, Inc., 2006.

[2] Michael Barr. Embedded Systems Glossary. http://www.netrino.com/Embedded-Systems/Glossary.

[3] Robert N. Charette. This Car Runs on Code. *Spectrum, IEEE*, 46(2), 2009. http://spectrum.ieee.org/green-tech/advanced-cars/this-car-runs-on-code.

[4] Thomas A. Henzinger and Joseph Sifakis. The Embedded Systems Design Challenge. In *Proceedings of the 14th International Symposium on Formal Methods (FM), Lecture Notes in Computer Science*, pages 1–15. Springer, 2006.

[5] Ivica Crnkovic and Magnus Larsson. *Building Reliable Component-Based Software Systems*. Artech House, Inc., Norwood, MA, USA, 2002.

[6] AUTOSAR Techincal Overview, Version 2.2.2. AUTOSAR – AUTomotive Open System ARchitecture, Release 3.1, The AUTOSAR Consortium, Aug., 2008. http://autosar.org.

[7] M. Broy, I.H. Kruger, A. Pretschner, and C. Salzmann. Engineering automotive software. *Proceedings of the IEEE*, 95(2):356 –373, feb. 2007.

[8] Ken Tindell and John Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocess. Microprogram.*, 40:117–134, April 1994.

[9] Robert Bosch GmbH. CAN Specification Version 2.0. Postfach 30 02 40, D-70442 Stuttgart, 1991.

[10] ISO 11898-1. Road Vehicles interchange of digital information controller area network (CAN) for high-speed communication, ISO Standard-11898, Nov. 1993.

[11] Automotive networks. CAN in Automation (CiA). http://www.can-cia.org/index.php?id=416.

[12] CAL, CAN Application Layer for Industrial Applications, CiA Draft Standard DS-207, Version 1.1. *CAN-in-Automation*, Feb. 1996.

[13] CANopen high-level protocol for CAN-bus, Version 3.0. *NIKHEF, Amsterdam*, March 2000. http://www.nikhef.nl/pub/departments/ct/po/doc/CANopen.pdf.

[14] Jimmy Westerlund. Hägglunds Controller Area Network (HCAN), Network Implementation Specification. *BAE Systems Hägglunds, Sweden (internal document)*, April 2009.

[15] MilCAN (CAN for Military Land Systems domain). http://www.milcan.org/.

[16] TIMMO Methodology , Version 2. *TIMMO (TIMing MOdel), Deliverable 7*, October 2009. The TIMMO Consortium.

[17] K. Hänninen et.al. The Rubus Component Model for Resource Constrained Real-Time Systems. In *3rd IEEE International Symposium on Industrial Embedded Systems*, June 2008.

[18] Sverine Sentilles, Aneta Vulgarakis, Tomas Bures, Jan Carlson, and Ivica Crnkovic. A Component Model for Control-Intensive Distributed Embedded Systems. In *Proceedings of the 11th International Symposium on Component Based Software Engineering (CBSE2008)*, pages 310–317. Springer Berlin, October 2008.

[19] Xu Ke, K. Sierszecki, and C. Angelov. COMDES-II: A Component-Based Framework for Generative Development of Distributed Real-Time Control Systems. In *Embedded and Real-Time Computing Systems and Applications, RTCSA 2007. 13th IEEE International Conference on*, pages 199 –208, August 2007.

[20] Arcticus Systems. http://www.arcticus-systems.com.

[21] EAST-ADL Domain Model Specification, Deliverable D4.1.1. http://www.atesst.org/home/liblocal/docs/ATESST2_D4.1.1_EAST-ADL 2-Specification_2010-06-02.pdf.

[22] Simulink - Simulation and Model-Based Design. http://www.mathworks .se/products/simulink.

[23] Rational Statemate. http://www-01.ibm.com/software/awdtools/statemate.

[24] TADL: Timing Augmented Description Language, Version 2. *TIMMO (TIMing MOdel), Deliverable 6*, October 2009. The TIMMO Consortium.

[25] Robert I. Davis, Steffen Kollmann, Victor Pollex, and Frank Slomka. Controller Area Network (CAN) Schedulability Analysis with FIFO queues. In *23rd Euromicro Conference on Real-Time Systems (ECRTS11)*, July 2011.

[26] Mubeen, Saad and Mäki-Turja, Jukka and Sjödin, Mikael. Extending response-time analysis of controller area network (CAN) with FIFO queues for mixed messages. In *Emerging Technologies Factory Automation (ETFA), IEEE 16th Conference on*, pages 1–4, sept. 2011.

[27] N. Feiertag, K. Richter, J. Nordlander, and J. Jonsson. A Compositional Framework for End-to-End Path Delay Calculation of Automotive Systems under Different Path Semantics. In *Compositional Theory and Technology for Real-Time Embedded Systems, 2008. CRTS 2008. Workshop on*, dec. 2008.

# II

# Included Papers

# Chapter 4

# Paper A: Analyzable Modeling of Legacy Communication in Component-Based Distributed Embedded Systems

Saad Mubeen, Jukka Mäki-Turja, Mikael Sjödin and Jan Carlson.
In proceedings of the $37^{th}$ Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pages 229-238, Oulu, Finland, September, 2011.

**Abstract**

We present extensions to the existing industrial component model Rubus Component Model (RCM). By introducing special purpose components to encapsulate and abstract the communication protocols in distributed embedded systems we allow use of legacy nodes and legacy protocols in a component-based and model-based software engineering environment. With the addition of these components, RCM will be able to support state-of-the-practice development processes for distributed embedded systems where communication rules are defined early in the development process. The proposed extension also allows model-based and component-based development of new nodes that are deployed in the legacy systems that use predefined communication rules. We also demonstrate how an end-to-end timing model can be extracted from a distributed embedded system modeled with extended RCM. The extracted model is then used to perform an end-to-end timing analysis that we implemented in the Rubus Analysis Framework.

## 4.1 Introduction

Embedded systems are found in almost all electronic products around us. Their applications span over many domains including automotive, aerospace, consumer electronics, biomedical, military applications, business applications, industrial control, etc. It is claimed in [1] that more than 98 percent of the processors produced today are embedded processors. Not only the number of embedded processors has increased in the past few years but the software which runs on them, i.e., the embedded software has also drastically increased in size and complexity. In automotive domain, for example, a modern premium car contains nearly 100 million lines of code that run on about 70 to 100 embedded processors [2]. Because of the continuously increasing trend in size and complexity of embedded software, the development of embedded systems has become very complex.

Often, embedded systems are resource-constrained and have hard real-time requirements. In order to capture such requirements as early as possible during the process of system development, handle complexity of embedded software, lower development cost, reduce time-to-market and time-to-test, allow reusability and modeling at higher level of abstraction, etc., the research community proposed the use of Model-Based Engineering (MBE) and Component-Based Software Engineering (CBSE) for the development of embedded systems [3, 4]. MBE provides the means to use models throughout the process of system development while CBSE facilitates the development of large software systems by integration of software components. CBSE raises the level of abstraction for software development and makes it possible to reuse software components and their architectures. There is a great need for bringing these development techniques in the embedded systems industry.

In distributed embedded systems, the functionality is distributed over many nodes and the nodes communicate with each other through a bus or a network. Software development of distributed embedded systems is more complex compared to single processor embedded systems. When MBE and CBSE are used for the development of resource-constrained and hard real-time distributed embedded systems, modeling of communication infrastructure arises as another challenge. In the industry, embedded systems are often deployed in legacy systems (previously developed) which use predefined rules for communication. Furthermore, distributed embedded systems are often expected to use legacy network protocols for real-time communication. A component model for the development of distributed embedded systems should not only be resource-efficient, but also abstract the application software from the communication

infrastructure. Moreover, it should support the modeling of legacy communications and legacy systems.

In this paper we propose an extension to a commercially available component model, the Rubus Component Model (RCM) [5], used for the development of resource-constrained real-time embedded systems in many domains especially automotive. It supports glue code generation, end-to-end response-time analysis, and resource requirement estimations. Over the years, RCM has evolved based on the industrial needs and the state-of-the-art research results. At present, RCM is able to model only single-node embedded systems. We extend RCM by adding special purpose components to it. The purpose of new components is to encapsulate and abstract the communication protocols and configuration in a component-based and model-based software engineering setting. The motivation for the extension of RCM comes from the industrial demand to model distributed embedded systems, real-time network communication, legacy communications and legacy systems.

### 4.1.1 Goals and Paper Contributions

We present an extension to an existing industrial component model by introducing new components to it. Our main goals in introducing new components are:

1. Allow model-based and component-based development of new nodes that are deployed in legacy systems that use predefined communication rules.

2. Support state-of-the-practice development processes where communication rules are defined early in the development process.

3. Enable adaptation of a node when communication rules change (e.g. due to re-deployment in a new system or due to upgrades in the communication system) without affecting the internal component design.

4. Generate these special components from the information about legacy communication or from early design decisions about network communication. The generated components should be compatible with the existing entities defining functionality and communication in RCM.

These goals are to be realized in RCM. The scope of this paper is PSMs (Platform Specific Models) for distributed embedded systems. With PSM we mean that the software components have been allocated to nodes and any adaptation

to specific node characteristics (e.g., device drivers and memory layouts) has been added to the model. Using our new components, nodes can be developed without explicit knowledge about the communication configuration.

One important objective during the extension of RCM is to enable the developer to specify real-time properties and analyze timing behavior of the modeled distributed embedded system. While making design decisions about the new modeling concepts and components, we placed special focus on how the modeled system will render itself to an end-to-end timing analysis. In this paper, we also show how we extract an end-to-end timing model from a distributed embedded system using the Rubus tool suite. The extracted model is then used to perform an end-to-end timing analysis that we implemented in the Rubus Analysis Framework [6, 7].

### 4.1.2 Paper Layout

The rest of the paper is organized as follows. Section 4.2 presents the Rubus concept, the component model and its development environment. In Section 4.3, we present the related research and compare different modeling approaches with ours. Section 4.4 describes the new modeling objects that support modeling of legacy communication. Section 4.5 describes the implementation of the end-to-end timing analysis of distributed embedded systems in the Rubus tool suite. Section 4.6 concludes the paper and presents the future work.

## 4.2 Background – The Rubus Concept

Rubus is a collection of methods and tools for model- and component-based development of dependable embedded real-time systems. Rubus is developed by Arcticus Systems [6] in close collaboration with several academic and industrial partners. Rubus is today mainly used for development of control functionality in vehicles. The Rubus concept is based around the Rubus Component Model (RCM) [5] and its development environment Rubus-ICE, which includes modeling tools, code generators, analysis tools and run-time infrastructure. The overall goal of Rubus is to be aggressively resource efficient and to provide means for developing predictable and analyzable control functions in resource-constrained embedded systems.

### 4.2.1 The Rubus Component Model

The purpose of the component model is to express the infrastructure for software functions i.e. the interaction between the software functions in terms of data and control flow. One important principle is to separate functional code and infrastructure implementing the execution model, i.e., explicit synchronization or data access should all be visible at the modeling level. In RCM, the basic component is called a Software Circuit (SWC). It is the lowest-level hierarchical element in RCM and its purpose is to encapsulate basic functions. The SWCs interact with each other through the use of ports. An SWC can be seen as a type, or a class, that can be instantiated an arbitrary number of times. By separating functional code and the infrastructure, RCM facilitates analysis and reuse of components in different contexts (an SWC has no knowledge how it connects to other components).

The execution semantics of software components (functions) is simply:

1. Upon triggering, read data on data in-ports.
2. Execute the function.
3. Write data on data out-ports.
4. Activate the output trigger.

An example system modeled with RCM, depicted in Figure 4.1, shows how components interact with external events and actuators with regard to both data and triggering. The triggering events can consist of interrupts, internal periodic clocks, internal and external events. Furthermore, the component model has a possibility to encapsulate SWCs into software assemblies enabling the designer to construct the system at different hierarchical levels.

### 4.2.2 The Rubus Code Generator and Run-Time System

From the resulting architecture of connected SWCs, functions are mapped to run-time entities; tasks. Each external event trigger defines a task and SWCs connected through the chain of triggered SWCs (triggering chain) are allocated to the corresponding task. All clock triggered "chains" are allocated to an automatically generated static schedule that fulfills the precedence order and temporal requirements.

Within trigger chains, inter-SWC communication is aggressively optimized to use the most efficient means of communication possible for each communication link. For example, there is no use of semaphores in point-to-point

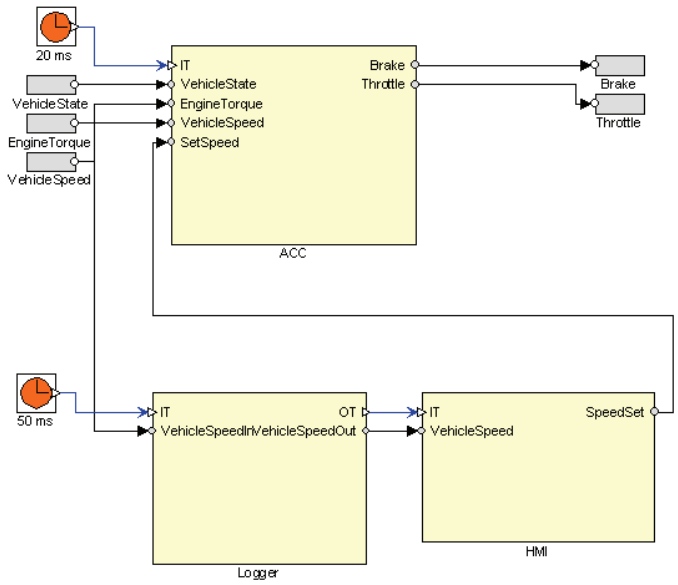


Figure 4.1: An example system in RCM

communications within a trigger chain. Another example is sharing of memory buffers between ports when there are no overlapping activation periods. This means that a buffer can be shared between two ports belonging to different SWCs if it can be guaranteed that these ports will never use the buffer space at the same time. This is true in the case of a trigger chain because a task early in the chain can never be active at the same time as a task late in the chain (considering the deadlines of tasks are smaller than their respective periods).

Allocation of SWCs to tasks and construction of schedule can be submitted to different optimization criterion to minimize, e.g., response times for different types of tasks, or memory usage. The run-time system executes all tasks on a shared stack, thus eliminating the need for static allocation of stack memory to each individual task.

### 4.2.3 The Rubus Analysis Framework

The model also allows expressing real-time requirements and properties on the architectural level. For example, it is possible to declare real-time requirements

from a generated event and an arbitrary output trigger along the trigger chain. For this purpose, the designer has to express real-time properties of SWCs, such as worst-case execution times and stack usage. The scheduler will take these real-time constraints into consideration when producing a schedule. For event-triggered tasks, response-time calculations are performed and compared to the requirements.

## 4.3 Related Work

There exist many component models for the development of distributed systems, e.g., Distributed Component Object Model (DCOM) [8], Common Object Request Broker Architecture (CORBA) [9], Enterprise JavaBeans (EJB) [10], etc. These models in their original form are not suitable for the development of resource-constrained distributed embedded systems with hard real-time requirements because they require excessive amount of computing resources, have large memory foot print and have inadequate support for modeling of real-time communication.

There are very few commercial component models for the development of distributed embedded and real-time systems especially in automotive domain. In the last decade, automotive research community and industry has focused more on the component-based development of automotive embedded systems which led to the development of various solutions, approaches, methodologies, and models. Some of them are discussed below.

### 4.3.1 AUTOSAR

AUTOSAR (AUTomotive Open System ARchitecture) [11] is a standardized software architecture for the development of software in automotive domain. It can be viewed as a standardized distributed component model [12]. In AUTOSAR, the application software is defined in terms of Software Components (SWCs). The distribution of SWCs, their virtual integration and communication at design time is handled by the Virtual Function Bus (VFB). The run-time representation of VFB for each Electronic Control Unit (ECU) is defined by the Run-Time Environment (RTE). The communication services are provided by the Basic Software (BSW) via RTE to the AUTOSAR SWCs.

When AUTOSAR was being developed, there was no focus placed on the specification and handling of real-time requirements and properties during the process of system development. On the other hand, such requirements and ca-

pabilities were strictly taken into account right from the beginning during the development of RCM. AUTOSAR describes embedded software development at a relatively higher level of abstraction compared to RCM. A Software Circuit in RCM more resembles to a runnable entity compared to AUTOSAR SWC. A runnable entity is a schedulable part of AUTOSAR SWC. As compared to AUTOSAR, RCM clearly distinguishes between the control flow and the data flow among SWCs in a node. AUTOSAR hides the modeling of execution environment. On the other hand, RCM explicitly allows the modeling of execution requirements, e.g., jitter, deadlines, etc., at an abstraction level close to the functional modeling while abstracting the implementation details.

In RCM, special Software Circuits, which are integral part of our contribution in this paper and will be introduced in the next Section, are used if SWCs require inter-ECU communication; otherwise, SWCs communicate via data and trigger ports. On the other hand, AUTOSAR does not differentiate between intra-node and inter-node communication at modeling level. Unlike RCM, there are no special components in AUTOSAR for inter-node communication. AUTOSAR SWCs use interfaces for all types of communications which can be of two types, i.e., Sender Receiver and Client Server. The Sender Receiver communication mechanism in AUTOSAR is very similar to the pipe-and-filter communication mechanism for component interconnection used in RCM.

### 4.3.2 TIMMO

TIMMO (TIMing MOdel) [13] is an initiative to provide AUTOSAR with a timing model. It describes a predictable methodology and a language, TADL (Timing Augmented Description Language) [14], to express timing requirements and timing constraints during all design phases in the development of automotive embedded systems. TADL is inspired by MARTE (Modeling and Analysis of Real Time and Embedded systems) [15] which is a UML profile for model-driven development of real-time and embedded systems. TIMMO development methodology makes use of structural modeling provided by EAST-ADL [16] which is a domain specific architecture description language used in automotive domain. TIMMO methodology and its model structure abstract the modeling of communication at implementation level of EAST-ADL where they propose to use AUTOSAR. Both TIMMO methodology and TADL have been evaluated on prototype validators. To the best of our knowledge there is no concrete industrial implementation of TIMMO. In TIMMO-2-USE project [17], the results of TIMMO will be further validated and brought to the indus-

try.

### 4.3.3 ProCom

ProCom [18] is a two-layer component model for the development of distributed embedded systems. At the upper layer, called ProSys, it models a system with concurrent subsystems which communicate by passing messages via explicit message channels. Unlike an RCM SWC, a subsystem is active which means that it has its own thread of execution and hence, it can be triggered periodically or by internal events. At the lower layer, called ProSave, a subsystem is internally modeled in terms of functional components which are implemented as a piece of code, for example, a C function. Like RCM SWCs, ProSave components are passive which means that they cannot trigger themselves and hence, they require an external trigger for activation.

ProCom is inspired by RCM, and there are a number of similarities between the ProSave modeling layer and RCM. For example, components in both ProSave and RCM are passive. Similarly, both the models clearly separate data flow from control flow among the components. Moreover, the communication mechanism for component interconnection used in both the models is pipe-and-filter. At modeling level, ProCom does not differentiate between inter-node and intra-node communication. ProCom uses two-step deployment modeling, i.e., virtual node modeling and physical node modeling [19]. At present, physical node modeling is a work in progress. The validation of a complete distributed embedded system, modeled with ProCom, is yet to be done. Moreover, the development environment and the tools accompanying ProCom are still evolving.

### 4.3.4 COMDES-II

COMDES-II (COMponent-based design of software for Distributed Embedded Systems) provides a component-based framework for the development of distributed embedded control systems [20]. It models the architecture of a system at two levels. At upper level, an application is modeled as a network of actors that are active components. Actors communicate with each other by sending labeled messages. At the lower level, the functionality of an actor is modeled in terms of Function Blocks which are passive components similar to the SWCs in RCM. The Operating System (OS) employed by COMDES-II implements fixed-priority timed multitasking scheduling. On the other hand, Rubus OS implements hybrid scheduling policy that combines both static cyclic scheduling and fixed-priority preemptive scheduling [21]. COMDES-II is a relatively

new research project and the support for development tools and run-time environment was provided fairly recently [22]. On the other hand, RCM and its tool suite are relatively mature as they are being used in the industry for the development of embedded systems for more than 10 years [6].

### 4.3.5 Real-Time CORBA

Object Management Group (OMG) defined middleware technologies such as Real-Time CORBA, minimum CORBA and CORBA lightweight services for the development of real-time and distributed embedded systems [23]. In some projects, Real-Time CORBA has been used to develop distributed embedded and real-time systems [24, 25]. Because of higher resource requirements, these models may not be suitable for the development of resource-constrained distributed embedded systems with hard real-time requirements.

### 4.3.6 Discussion

RCM can be considered a suitable choice for the development of resource-constrained distributed embedded systems for many reasons. For example, it can completely handle timing related information, i.e., real-time requirements, properties and constraints during all the stages of system development; It has a small run-time footprint (timing and memory overhead); it implements the state-of-the-art research results; It has a strong support for development and analysis tools, etc.

## 4.4 Support for Modeling of Legacy Communication

In an ideal scenario, it should be possible to automatically generate the communication from the design model for each distributed embedded application. However, this is often not the practice in the industry because of legacy communications and legacy systems. These systems have their own predefined rules of communication. Our goal is to introduce the support for modeling of legacy communications in RCM.

To support abstraction of the implementation of communications in a node, we propose the introduction of two special purpose modeling elements, i.e., Output Software Circuit (OSWC) and Input Software Circuit (ISWC) for each

frame that is to be sent or received by a node (connected to a network) respectively. In order to represent a model of communication in a physical bus, we propose another modeling object called Network Specification (NS).

### 4.4.1 Network Specification (NS)

It is the model representation of a physical bus. It consists of two parts: one is protocol independent and the other is protocol dependent. The protocol-independent part defines messages and the data-elements mapped to these messages. Moreover, it describes message properties, i.e., a message ID, a unique sender node ID, a list of receiver nodes IDs and an ordered set of RCM signals. A signal in RCM has a name, data type and real-time properties.

The protocol-dependent part of NS defines the behavior semantics of each message according to the protocol used for network communication. It is specific to each protocol, e.g., it will be different for CANopen [26], Hägglunds Controller Area Network (HCAN) [27], MilCAN (CAN for Military Land Systems domain) [28], Flexray, etc. The protocol-dependent part of NS contains complete information of all the frames which are sent on the bus. Moreover, it describes the frame properties. In RCM, a frame is a collection of RCM signals.

The frame properties described by the protocol-dependent part of NS (e.g., for a CANopen protocol) include an identifier (a reference to the corresponding message in the protocol-independent part of NS), a priority, a transmission type (type of message transmission in CANopen), a sender node ID, a list of receiver nodes IDs, whether a frame is an IN frame or an OUT frame, a period (period with which a message is sent in case of periodic transmission), an inhibit time (minimum time between successive transmission of a message in case of one of the asynchronous transmission types in CANopen), SYNC period (time between SYNC messages sent by the CANopen SYNC master), and real-time requirements. Moreover, it also specifies the speed of CAN bus. The transmission type of a frame can be periodic, event or mixed (transmitted periodically as well as on arrival of an event) [29].

The components inside a single node communicate with each other by using data and control signals separately. However, if a component on one node intends to communicate with a component on another node via a network then the signals are packed into frames. These frames are then transmitted over the network. Here, some questions arise regarding the network communication. How are signals packed into the frames? How many signals a message contains? How are signals encoded into the frames at the sender

node? How are signals decoded from the received frames and sent to the respective SWCs at the receiver node? All the rules that are concerned with the answers to these questions are specified in the Signal Mapping. The Signal Mapping is unique for each network communication protocol and is defined by the protocol-dependent part of NS.

### 4.4.2 Output Software Circuit (OSWC)

OSWC is the model representation of signals in an outgoing message (frame) to the network. Basically, it is a Software Circuit which denotes the data that leaves the model. An OSWC is associated with a LAN (Local Area Network) object. In RCM, LAN is an object to represent a connection between two or more nodes in a system. Formally, a LAN is defined by its name, a list of connected nodes and a Network Specification. There is exactly one OSWC in a node for every outgoing frame on the network. Each OSWC describes all the signals that can be sent in a particular frame. A frame contains zero or more signals.

An OSWC has only one trigger in-port and at least one data in-port. Each data in-port is associated with one signal in the Network Specification. Therefore, the number of data in-ports may vary depending upon the number of signals to be packed in the frame. An OSWC has no data and trigger out-ports. The OSWC component uses protocol-specific rules, specified in the protocol-specific part of NS, while encoding data and mapping signals to a frame. In this way, OSWC provides a clear abstraction to the SWCs that send signals to one of its data in-ports. Thus, SWCs are kept unaware of the protocol-specific details such as signal-to-frame mapping, data type encoding and transmission patterns of frames. The OSWC component is graphically illustrated in Figure 4.2.

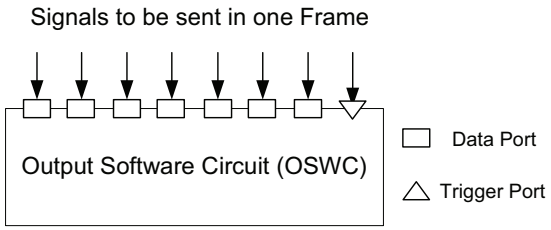


Figure 4.2: Graphical illustration of OSWC.

### 4.4.3 Input Software Circuit (ISWC)

ISWC is the model representation of signals in an incoming message (frame) from the network. Basically, it is a Software Circuit which denotes the data that enters the model. An ISWC is associated with a LAN object defined in RCM. There is exactly one ISWC component in a node for every frame received from the network. An ISWC describes all the signals that are contained in a received frame associated to it. An ISWC has one unconditional trigger out-port. An unconditional trigger port produces a trigger signal every time the SWC is executed. There is at least one data out-port in the ISWC component. Each data out-port is associated with one signal in the Network Specification of the LAN object. Therefore, the number of data out-ports may vary depending upon the number of signals contained in the received frame. An ISWC has no data out-ports. There is one trigger in-port in every ISWC component which is triggered when a frame arrives from the network. When a frame arrives at a node, the physical bus drivers and protocol-specific implementation of ISWC extract the signals (zero or more signals per frame) and encode their data in the RCM data type. When the signal(s) is delivered, the data is placed on the data port which is connected to the data in-port of the destination SWC (the tracing information is provided in NS), and the corresponding trigger port is triggered. Figure 4.3 graphically illustrates ISWC.

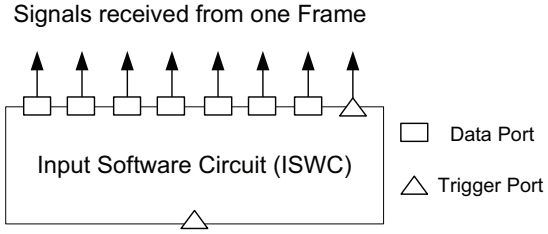


Figure 4.3: Graphical illustration of ISWC.

**Example**

Consider an example of a node in a distributed embedded application modeled with newly introduced objects in RCM as shown in Figure 4.4. The network protocol considered in this example is CAN. Note that the figure is divided into two halves: the upper half represents the model of a node whereas the lower half depicts the physical communication including CAN controller

and CAN network. There are two grey boxes outside the model called CAN SEND and CAN RECEIVE that are placed just below the sets of OSWCs and ISWCs respectively. These gray boxes are specific for each network protocol. The frames that leave the model (sent to CAN SEND) are denoted by $S$ (Send), e.g., $S1$, $S2$ and $S3$. Similarly all the frames that enter the model (received from CAN RECEIVE) are denoted by $R$ (Receive), e.g., $R1$ and $R2$ as shown in Figure 4.4.

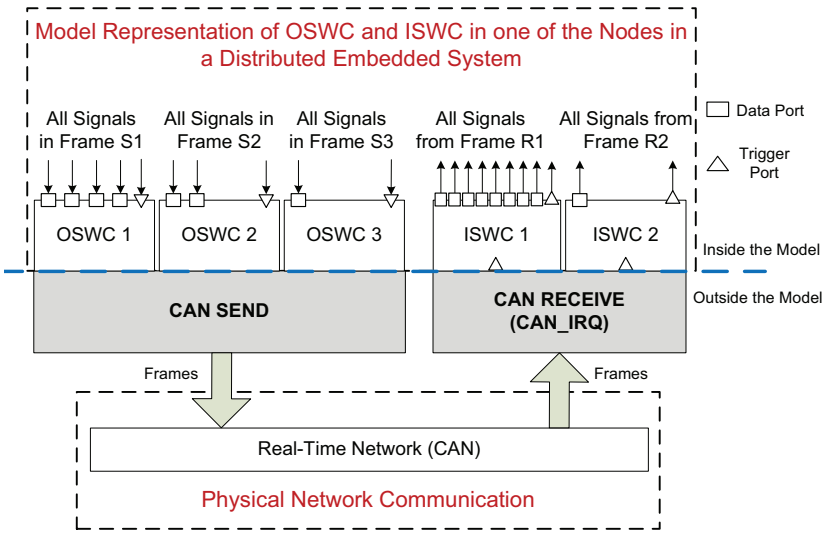


Figure 4.4: Model of OSWCs and ISWCs in one of the nodes in a distributed embedded system modeled with extended RCM.

All the signals sent in the frame $S1$ are provided at the data in-ports of OSWC1. These signals are mapped and encoded into $S1$ by OSWC1 according to the protocol-specific information available in the Network Specification. Once the frame is ready, it leaves the model as it is sent to the grey box CAN SEND. In this example, this grey box represents a CAN controller in the node which is responsible for the physical transmission of this frame on the network according to the communication rules of CAN protocol.

When a frame arrives at the receiving node, it is transferred by the physical network drivers to a grey box (CAN RECEIVE in this example) that produces an interrupt. The frame enters the model and is transferred to the destina-

tion ISWC (the tracing information is provided in the Network Specification). ISWC extracts the signals from the frame, decodes the data from the frame and encodes it to RCM data type. The data is placed on the data out-port of ISWC which is connected to the data in-port of the destination SWC and the corresponding trigger out-port is triggered (the tracing information is provided in the Network Specification).

### 4.4.4 Automatic Generation of OSWC and ISWC

Both OSWC and ISWC can be automatically generated from NS by a Network Configuration Tool. The input to this tool is the protocol-specific information about the network communication and the tracing information of tasks in all the distributed transactions (event-based and periodic chains) present in the application. This information is made available from the configuration files that correspond to the NS. The output of this tool is a set of automatically generated OSWCs and ISWCs for each node in the network. This tool also carries out mapping from NS to OSWC and ISWC and vice versa. The Input and Output Software Circuits are translated into a set of SWCs to execute the protocol at run-time.

### 4.4.5 Discussion

Let us briefly compare our newly introduced modeling approach for network communication with the existing modeling approach for intra-node communication in RCM (depicted in Figure 4.1 by means of connectors). An alternative to the new modeling approach (presented in this paper) would have been to use the same connectors for modeling both inter-node and intra-node communication by attaching a boolean specifier to it, say, 0 for intra-node and 1 for inter-node communication; and some tool could automagically generate the run-time architecture for all communications and perform the deployment of the distributed embedded application. Similarly, another alternative modeling approach is to have an allocation property on each SWC, describing which node it will run on. However, these modeling approaches may not be practical in an industrial setup because of the requirements of modeling legacy systems and legacy communications, deployment of newly developed nodes in the existing systems and early analysis of the developed system.

Analyzability was one important aspect that was kept in mind while introducing new components in RCM. The objective was to enable RCM to not only model the legacy communication but also to analyze the end-to-end timing be-

havior of the modeled system. In the next Section, we will discuss how the required timing information is extracted from a distributed embedded system, modeled with RCM, to perform an end-to-end timing analysis.

## 4.5 Implementation of End-to-End Timing Analysis in Rubus-ICE

In real-time systems, the time at which the result is available is as important as correct value of the result. With the newly introduced modeling elements in RCM, we can model a complete distributed real-time embedded system. In order to ensure that all timing requirements are met, the modeled system should render itself to an end-to-end timing analysis. To perform the timing analysis, an end-to-end timing model of the application should be available. In this Section, we first present the end-to-end timing model used by the Rubus Analysis Framework. Then we demonstrate, by an example, the extraction of the end-to-end timing model. Finally, we describe the support for the end-to-end timing analysis available in Rubus-ICE.

### 4.5.1 System Model for End-to-end Timing Analysis

The scheduling model that we implemented in the Rubus Analysis Framework, to carry out the end-to-end timing analysis, consists of two state-of-the-art scheduling models, i.e., a node analysis model and a network analysis model. The node analysis model was previously implemented in the the analysis framework of Rubus-ICE [7]. The node analysis model is a task model that corresponds to the tighter version of offset-based RTA [30] that is adapted from the scheduling model for the holistic response-time analysis developed by [31] and later on, extended by many researchers, e.g., [32, 33]. This model is used for the response-time analysis of tasks in a node. The network analysis model that we implemented in Rubus-ICE is a communication model [34] which is used for the response-time analysis of CAN messages. The task model and the communication model together comprise the end-to-end timing model of a distributed real-time and embedded system.

#### Node Analysis Model

The system (node), $\Gamma$, consists of a set of $k$ transactions $\Gamma_1, \ldots, \Gamma_k$. Each transaction $\Gamma_i$ is activated by a periodic sequence of events with a period $T_i$.

In case of sporadic events, $T_i$ denotes the minimum inter-arrival time between two consecutive events. In this model we consider that the activating events are mutually independent, i.e., the phasing between them is arbitrary. There are $|\Gamma_i|$ tasks in a transaction $\Gamma_i$ and each task may not be activated until a certain time, called an *offset*, elapses after the arrival of the external event. By task activation we mean that the task is released for execution. A task is denoted by $\tau_{ij}$. The first subscript, *i*, specifies the transaction to which this task belongs and the second subscript, *j*, denotes the number of the task within the transaction. A task, $\tau_{ij}$, is defined by the following attributes.

- $C_{ij}$: It is the worst case execution time of the task.
- $O_{ij}$: It is an offset of the task.
- $D_{ij}$: It is the deadline of the task.
- $J_{ij}$: It is the maximum release jitter.
- $B_{ij}$: It represents the maximum blocking of the task from lower priority tasks.
- $P_{ij}$: It represents the priority of the task.
- $R_{ij}$: It represents the worst-case response time of the task.

In this task model, there are no restrictions placed on offset, deadline or jitter, i.e., they can each be either smaller or greater than the period.

**Network Analysis Model**

The system (network) consists of a number of nodes that are connected through a CAN bus. If a task on one node intends to communicate with a task on another node, it queues a message in the send queue of its node. The CAN protocol ensures arbitration and transmission of all messages over the bus. There are four different types of CAN frames used for message transmission, i.e., Data frame, Remote Transmit Request (RTR) frame, Overload frame and Error frame. In this model, a message corresponds to a message that uses Data or RTR frames for transmission. Each message $m$ has the following attributes.

- $ID_m$: It is a unique identifier.
- $FRAME\_TYPE$: It specifies whether the frame is a Standard or an Extended CAN frame.

- $TRANSMISSION\_TYPE$: It specifies whether the frame is periodic or event or mixed (both periodic and event).
- $P_m$: It is a unique priority.
- $C_m$: It specifies the transmission time of the message.
- $J_m$: It is a release jitter that is inherited from the response time of the task queueing the message.
- $DLC_m$: Each message can carry a data payload that ranges from 0 to 8 bytes. This number is specified in the header field of the frame called Data Length Code.
- $T_m$: It specifies the period of a message in case of periodic transmission. In case of an event transmission, $T_m$ refers to the minimum time that should elapse between the transmission of any two messages.
- $B_m$: Each message has a blocking time which refers to the maximum amount of time this message can be blocked by the lower priority messages.
- $R_m$: It denotes the worst-case response time of a message $m$.

### 4.5.2 Extraction of End-to-End Timing Model

We extract an end-to-end timing model from the distributed transactions modeled with extended RCM. The extracted model is used to analyze the end-to-end timing for delays and network utilization. Consider the following example.

**Example**

An example distributed embedded system modeled with RCM using the new modeling objects is shown in Figure 4.5. There are two nodes in the system with three SWCs per node. SWCs communicate with each other by using both inter-node and intra-node communication. For inter-node communication, CAN or any high level protocol of CAN (e.g., CANopen, HCAN, MilCAN, etc.) can be used. In this example, the nodes are connected to a CAN network. An event chain (distributed transaction) that consists of four Software Circuits, i.e., SWC1, SWC2, SWC4 and SWC5 is identified with bold lines in Figure 4.5. In this transaction, a clock triggers SWC1 which in turn triggers SWC2. SWC2 then sends a signal to the OSWC A1 which in turn maps it to a CAN frame. It then sends the frame to the grey box CAN SEND and hence,

the data leaves the model. The frame is transmitted over the CAN bus by the CAN controller according to the communication rules of CAN protocol.

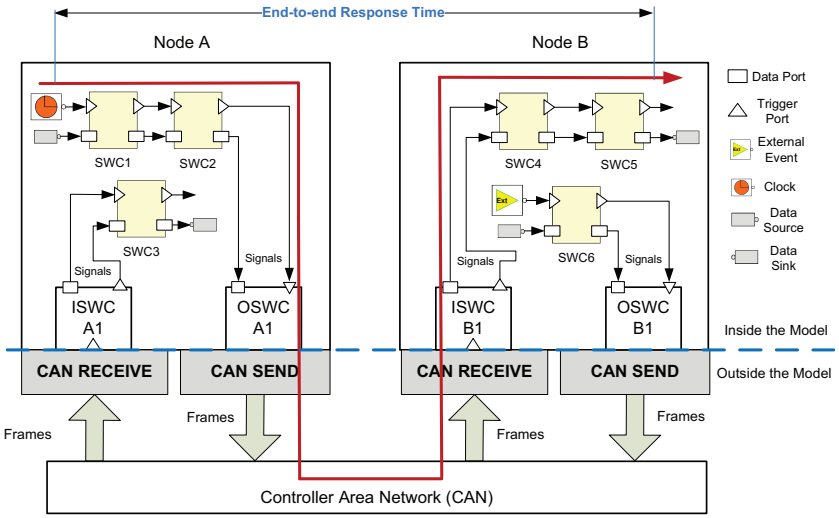


Figure 4.5: Example distributed system modeled with extended RCM

When the frame is received at Node B, the grey box CAN RECEIVE raises an interrupt and passes the frame to ISWC B1 and hence, the data enters the model. It should be noted that there can be more than one ISWCs in a node. In that case, a received frame is passed to the desired ISWC by looking at the tracing information in the NS. The ISWC $B1$ decodes the received frame, extracts signal from the frame, places the data on the corresponding data port and triggers the corresponding trigger port. The elapsed time between the arrival of a triggering event (clock trigger) at the input of SWC1 and the response of SWC5 is referred to as an end-to-end response time of the distributed transaction and is indicated in Figure 4.5.

The end-to-end timing model, used by the Rubus Analysis Framework, requires the timing related information of all the transactions and messages as discussed in the above subsection. The required timing information about all the transactions in the system is extracted from the compiled and verified design representation of the modeled systems provided in the form of node configuration files in Rubus-ICE. At network level, the timing information is specified in the Signal Mapping that is defined in NS. This information is ex-

tracted from the configuration specification files corresponding to NS.

### 4.5.3 Support for End-to-End Timing Analysis

In Rubus-ICE, when the designed model is completed it is compiled to the Intermediate Compiled Component Model (ICCM) file [7]. All the timing information required by the end-to-end timing model is extracted from the ICCM file. From this timing model, the Rubus Analysis Framework performs the response-time analysis of individual tasks [33], the response-time analysis of messages on the network [34, 35] and the end-to-end timing analysis [36]. The analysis framework provides the results, i.e., response times of individual tasks, response times of network messages, end-to-end response times of event chains (distributed transactions), network utilization, etc., back to Rubus-ICE. This whole process is depicted in Figure 4.6.

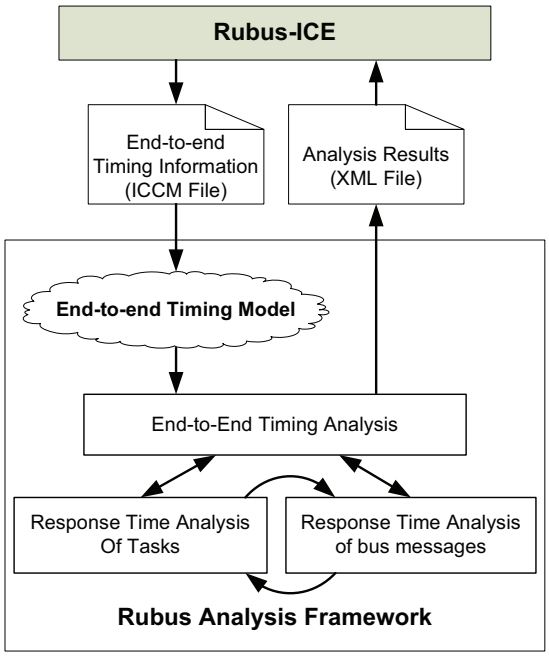


Figure 4.6: Extraction of end-to-end timing model for timing analysis in Rubus-ICE

## 4.6 Conclusion

We introduced new components to the industrial component model, the Rubus Component Model (RCM), for the development of distributed embedded systems. The purpose of new modeling objects, i.e., Output Software Circuit (OSWC), Input Software Circuit (ISWC) and Network Specification (NS) is to abstract the implementation and configuration of communications in distributed embedded systems. These objects make the communication capabilities of a node very explicit, but efficiently hide the implementation or protocol details. The extended model allows model-based and component-based development of new nodes that are deployed in legacy (previously developed) systems that use predefined communication rules. While making the design decisions about new components, one important objective was to enhance analyzability in the component model. Here, the focus was the ease to extract the end-to-end timing model from a distributed application modeled with RCM. The extracted model is used by the Rubus Analysis Framework to perform the end-to-end timing analysis.

With the addition of new modeling capabilities, RCM can be considered a suitable choice for the industrial development of resource-constrained distributed embedded systems with hard real-time requirements. There are a number of reasons behind this motivation, e.g., it can model real-time communication (both intra-node and inter-node); it can completely handle timing related information (real-time requirements, properties and constraints) during all the stages of system development; it has a small run-time footprint (timing and memory overhead); it implements the state-of-the-art research results; it has a strong support for development and analysis tools, etc.

In the future work, the implementation of OSWC and ISWC will be automatically generated from protocol configuration files of other specialized communication protocols used for real-time network communication such as CANopen, HCAN (Hägglunds Controller Area Network), J1939, etc. For example, the next step will be to generate automatically the implementation of OSWC and ISWC from DCFs (Device Configuration Files) in CANopen or for subsets of J1939.

## Acknowledgement

This work is supported by Swedish Knowledge Foundation (KKS) within the project EEMDEF, the Swedish Research Council (VR) within project TiPCES,

and the Strategic Research Foundation (SSF) with the centre PROGRESS. The authors would like to thank the industrial partners Arcticus Systems and BAE Systems Hägglunds for the cooperation.

# Bibliography


[1] M. Broy. Automotive software and systems engineering. In *Formal Methods and Models for Co-Design, 2005. MEMOCODE '05. Proceedings. Third ACM and IEEE International Conference on*, pages 143 – 149, 2005.

[2] Robert N. Charette. This Car Runs on Code. *Spectrum, IEEE*, 46(2), 2009. http://spectrum.ieee.org/green-tech/advanced-cars/this-car-runs-on-code.

[3] Thomas A. Henzinger and Joseph Sifakis. The Embedded Systems Design Challenge. In *Proceedings of the 14th International Symposium on Formal Methods (FM), Lecture Notes in Computer Science*, pages 1–15. Springer, 2006.

[4] Ivica Crnkovic and Magnus Larsson. *Building Reliable Component-Based Software Systems*. Artech House, Inc., Norwood, MA, USA, 2002.

[5] K. Hänninen et.al. The Rubus Component Model for Resource Constrained Real-Time Systems. In *3rd IEEE International Symposium on Industrial Embedded Systems*, June 2008.

[6] Arcticus Systems. http://www.arcticus-systems.com.

[7] K. Hänninen et.al. Framework for real-time analysis in Rubus-ICE. In *Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on*, pages 782 –788, 2008.

[8] Microsoft, Distributed Component Object Model (DCOM). http://msdn.microsoft.com/en-us/library/Aa286561.

[9] OMG, Common Object Request Broker Architecture (CORBA) , Version 3.1, January 2008.

[10] L. DeMichiel. Sun Microsystems, Enterprise JavaBeans Specification, Version 2.1. *Sun Microsystems*, 2002.

[11] AUTOSAR Techincal Overview, Version 2.2.2. AUTOSAR – AUTomotive Open System ARchitecture, Release 3.1, The AUTOSAR Consortium, Aug., 2008. http://autosar.org.

[12] Harald Heinecke et al. AUTOSAR – Current results and preparations for exploitation. In *Proceedings of the 7th Euroforum Conference*, EUROFORUM '06, May 2006.

[13] TIMMO Methodology , Version 2. *TIMMO (TIMing MOdel), Deliverable 7*, October 2009. The TIMMO Consortium.

[14] TADL: Timing Augmented Description Language, Version 2. *TIMMO (TIMing MOdel), Deliverable 6*, October 2009. The TIMMO Consortium.

[15] The UML Profile for MARTE: Modeling and Analysis of Real-Time and Embedded Systems, January 2010.

[16] EAST-ADL Domain Model Specification, Deliverable D4.1.1. http://www.atesst.org/home/liblocal/docs/ATESST2_D4.1.1_EAST-ADL 2-Specification_2010-06-02.pdf.

[17] TIMMO-2-USE. http://www.timmo-2-use.org/.

[18] Sverine Sentilles, Aneta Vulgarakis, Tomas Bures, Jan Carlson, and Ivica Crnkovic. A Component Model for Control-Intensive Distributed Embedded Systems. In *Proceedings of the 11th International Symposium on Component Based Software Engineering (CBSE2008)*, pages 310–317. Springer Berlin, October 2008.

[19] J. Carlson, J. Feljan, J. Mäki-Turja, and M. Sjödin. Deployment Modelling and Synthesis in a Component Model for Distributed Embedded Systems. In *36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), 2010*, pages 74 –82, 2010.

[20] Xu Ke, K. Sierszecki, and C. Angelov. COMDES-II: A Component-Based Framework for Generative Development of Distributed Real-Time Control Systems. In *Embedded and Real-Time Computing Systems and Applications, RTCSA 2007. 13th IEEE International Conference on*, pages 199 –208, August 2007.

[21] Jukka Mäki-Turja, Kaj Hänninen, and Mikael Nolin. Efficient Development of Real-Time Systems Using Hybrid Scheduling. In *9th Real-Time in Sweden (RTiS'07)*, pages 157–163, August 2007.

[22] Yu Guo, K. Sierszecki, and C. Angelov. COMDES Development Toolset. In *5th International Workshop on Formal Aspects of Component Software FACS 08, Malaga, Spain*, 2008.

[23] Catalog of Specialized CORBA Specifications. OMG Group. http://www.omg.org/technology/documents/.

[24] S. Lankes, A. Jabs, and T. Bernmerl. Integration of a CAN-based connection-oriented communication model into Real-Time CORBA. In *Parallel and Distributed Processing Symposium*, 2003.

[25] R. Finocchiaro, S. Lankes, and A. Jabs. Design of a real-time CORBA event service customised for the CAN bus. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, page 121, 2004.

[26] CANopen high-level protocol for CAN-bus, Version 3.0. *NIKHEF, Amsterdam*, March 2000. http://www.nikhef.nl/pub/departments/ct/po/doc/CANopen.pdf.

[27] Jimmy Westerlund. Hägglunds Controller Area Network (HCAN), Network Implementation Specification. *BAE Systems Hägglunds, Sweden (internal document)*, April 2009.

[28] MilCAN (CAN for Military Land Systems domain). http://www.milcan.org/.

[29] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Extending schedulability analysis of controller area network (CAN) for mixed (periodic/sporadic) messages. In *Emerging Technologies Factory Automation (ETFA), IEEE 16th Conference on*, sept. 2011.

[30] Jukka Mäki-Turja, , and Mikael Nolin. Tighter response-times for tasks with offsets. In *Real-time and Embedded Computing Systems and Applications Conference (RTCSA)*. Springer-Verlag, August 2004.

[31] Ken Tindell. Adding Time-Offsets to Schedulability Analysis. Technical report, Department of Computer Science, University of York, England, January 1994.

[32] J.C. Palencia and M. Gonzalez Harbour. Schedulability Analysis for Tasks with Static and Dynamic Offsets. *Real-Time Systems Symposium, IEEE International*, page 26, 1998.

[33] Jukka Mäki-Turja and Mikael Nolin. Efficient implementation of tight response-times for tasks with offsets. *Real-Time Syst.*, 40(1):77–116, 2008.

[34] K.W. Tindell, H. Hansson, and A.J. Wellings. Analysing real-time communications: controller area network (CAN). In *Real-Time Systems Symposium (RTSS) 1994*, pages 259 –263.

[35] Robert Davis, Alan Burns, Reinder Bril, and Johan Lukkien. Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems*, 35:239–272, 2007.

[36] Ken Tindell and John Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocess. Microprogram.*, 40:117–134, April 1994.

# Chapter 5

# Paper B: Extraction of End-to-end Timing Model from Component- Based Distributed Real-Time Embedded Systems

Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin.
In proceedings of the International Workshop on Time Analysis and Model-Based Design, from Functional Models to Distributed Deployments (TiMoBD) located at Embedded Systems Week, Taipei, Taiwan, October, 2011.

**Abstract**

In order to facilitate the end-to-end timing analysis early during the development of component-based distributed real-time embedded (DRE) systems, we present the extraction of end-to-end timing models using the existing industrial component model, Rubus Component Model (RCM). Moreover, we discuss and solve the issues involved during the model extraction such as, extraction of timing information from all nodes and networks in the system, tracing of event chains in distributed transactions, and modeling of exit and entry points for RCM models to provide timing bounds for extra-model medium. We also describe the implementation of end-to-end timing model extraction in the Rubus Analysis Framework.

## 5.1 Introduction

The model-based and component-based development [1, 2] is often considered a promising choice for the development of distributed real-time embedded (DRE) systems for many reasons such as: handling complexity of embedded software; lowering development cost; reducing time-to-market and time-to-test; allowing reusability; providing flexibility, maintainability and understandability; enabling modeling and analysis at higher level of abstraction and early during the process of system development, etc.

In DRE systems, the timing behavior of the system is as important as its functional behavior. The current trend for the industrial development of DRE systems, especially in automotive domain, is focused towards handling timing related information and performing timing analysis as early as possible during the process of system development [3, 4, 5]. This implies that the component model for the development of DRE systems should support the extraction of required timing information into an end-to-end timing model early during the development.

### 5.1.1 Goals and Paper Contribution

Our main goal is to extract an end-to-end timing model from component-based DRE system modeled with the existing industrial component model, i.e., the Rubus Component Model (RCM). We focus on the following issues.

1. To extract the timing information from all the nodes and networks in a DRE application into an end-to-end timing model.
2. To trace event chains in the transactions that are distributed over more than one node in a DRE system.
3. To model exit and entry points for RCM models to provide timing bounds for extra-model medium.
4. To implement the extraction of the end-to-end timing model in the Rubus Analysis Framework.

### 5.1.2 Paper Layout

The rest of the paper is organized as follows. In Section 5.2, we discuss the Rubus concept. Section 5.3 presents the related work. In Section 5.4, we

discuss main parts of an end-to-end timing model. In Section 5.5, we discuss the model extraction. Section 5.6 concludes the paper.

## 5.2 The Rubus Concept

Rubus is a collection of methods and tools for model-based development of dependable embedded real-time systems. The Rubus concept is based around the Rubus Component Model and its development environment Rubus-ICE (Integrated Component development Environment) [6, 7], which includes modeling tools, code generators, analysis tools and run-time infrastructure. The overall goal of Rubus is to be aggressively resource efficient and to provide means for developing predictable and analyzable control functions in resource-constrained embedded systems.

### 5.2.1 The Rubus Component Model (RCM)

RCM expresses the infrastructure for software functions, i.e., the interaction between the software functions in terms of data and control flow separately. The control flow is expressed by triggering objects as clocks and events as well as other components. In RCM, the basic component is called Software Circuit (SWC). It is the lowest-level hierarchical element in RCM. It is a self contained unit that encapsulates the basic functionality into manageable and reusable blocks of software. The execution semantics of an SWC is simply: upon triggering, read data on data in-ports; execute the function; write data on data out-ports; and activate the output trigger.

One important principle is to separate functional code and infrastructure implementing the execution model, i.e., explicit synchronization or data access should all be visible at the modeling level. By separating functional code and the infrastructure, RCM facilitates analysis and reuse of components in different contexts (an SWC has no knowledge how it connects to other components). The component model has the possibility to encapsulate SWCs into software assemblies enabling the designer to construct the system at different hierarchical levels.

### 5.2.2 The Rubus Code Generator and Run-Time System

From the resulting architecture of connected SWCs, functions are mapped to run-time entities; tasks. Each external event trigger defines a task and SWCs

connected through the chain of triggered SWCs (triggering chain) are allocated to the corresponding task. All clock triggered "chains" are allocated to an automatically generated static schedule that fulfills the precedence order and temporal requirements. Within trigger-chains, inter-SWC communication is aggressively optimized to use the most efficient means of communication possible for each communication link. Allocation of SWCs to tasks and construction of schedule can be submitted to different optimization criterion to minimize, e.g., response times for different types of tasks, or memory usage. The run-time system executes all tasks on a shared stack, thus eliminating the need for static allocation of stack memory to each individual task.

### 5.2.3 The Rubus Analysis Framework

The Rubus model allows expressing real-time requirements and properties at the architectural level. For example, it is possible to declare real-time requirements from a generated event and an arbitrary output trigger along the trigger chain. For this purpose, the designer has to express real-time properties of SWCs, such as worst-case execution times and stack usage. The scheduler will take these real-time constraints into consideration when producing a schedule. For event-triggered tasks, response-time calculations are performed and compared to the requirements. The analysis supported by the model includes distributed end-to-end response time analysis and shared stack analysis.

### 5.2.4 The Rubus Simulation Model

The Rubus SIMulation Model (RSIM) and accompanying tools enable simulation and testing of applications modeled with RCM at various hierarchical levels such as: an SWC or function, a hierarchical RCM component structure as an Assembly (ASM), a complete Electronic Control Unit (ECU) application (may require I/O simulation), a set of ECU's, a distributed system (may require I/O simulation of each ECU), etc. To verify the logical functionality of these objects, RSIM supports testing in an automatic generated framework based on the Rubus OS Simulator.

The input data is read from external tools or files, e.g., Matlab, and fed to the simulation process that controls the stimulation of input ports and state variables using probes. The output from the simulation process is fed back to the external tools. By building a simulated environment around the application to be simulated, the execution of the application can be controlled from a high-level tool such as LabView or Matlab/Simulink. The high-level tools control

the execution of the simulated target by means of commands to stop and run the target clock a specified number of ticks. The high-level tool sets the input data to the control function to be tested, performs a number of execution steps, and then reads the generated output data. In this way the execution flow can be visualized in each time increment.

## 5.3 Related Work

There are very few commercial component models for the development of DRE systems especially in automotive domain. In our previous work,we carried out a detailed comparison of RCM with various models for DRE systems [8]. We briefly highlight a few of them.

AUTOSAR (AUTomotive Open System ARchitecture) [9] is a standardized software architecture for the development of software in automotive domain. It can be viewed as a standardized distributed component model [10]. When AUTOSAR was being developed, there was no focus placed on the specification and handling of real-time requirements and properties during the process of system development. In the recent release of AUTOSAR 4.0, timing model is introduced and timing extensions such as timing constraints, requirements, budgets, event chains and event triggers are added[11].

TIMMO (TIMing MOdel) [5] is an initiative to provide AUTOSAR with a timing model. It describes a predictable methodology and a language, TADL (Timing Augmented Description Language) [4], to express timing requirements and timing constraints in all design phases during the development of automotive embedded systems. Both TIMMO methodology and TADL have been evaluated on prototype validators. To the best of our knowledge there is no concrete industrial implementation of TIMMO. In TIMMO-2-USE project [3], the results of TIMMO will be further validated and brought to the industry.

ProCom [12] is a two-layer component model for the development of distributed embedded systems. ProCom is inspired by RCM. The validation of a complete distributed embedded system, modeled with ProCom, is yet to be done. Moreover, the development environment and the tools accompanying ProCom are still evolving.

A related research presents a detailed overview of timing aspects during the design activities of automotive embedded systems[13]. In[14], the authors define end-to-end delay semantics and present a formal framework for the calculation of end-to-end delays for register-based multi-rate systems. Like any other timing analysis, they assume that the timing information of the system is

available as an input. On the other hand, our focus is on the extraction of required information into an end-to-end timing model to carry out the end-to-end timing analysis.

In our previous work, we extended RCM to model and analyze DRE systems. In [15], we explored various options for modeling of real-time network communication in RCM. In [16], we discussed our initial work to support modeling of legacy network communication in distributed embedded systems. In [8], we presented analyzable modeling of legacy communication in component-based DRE systems. We added new components in RCM to encapsulate and abstract the communication protocols, allow use of legacy nodes and legacy protocols in a component-based and model-based software engineering environment, and support model-based and component-based development of new nodes that are deployed in the legacy systems that use predefined communication rules. Further, we highlighted the problem of tracing event chains in the transactions that are distributed over several nodes in a DRE system [17].

## 5.4 End-to-end Timing Model

In this section, we discuss the main constituents of an end-to-end timing model of a DRE system. The end-to-end timing model consists of two models, i.e., system timing model and system tracing model. All the required timing information of each node in a DRE application is extracted into a node timing model. Similarly, the timing information of all the networks in a DRE application is extracted into a network timing model. Together the node and network timing model comprise the system timing model. All the mapping and tracing information of event chains and distributed transactions is extracted into a tracing model.

### 5.4.1 System Timing Model

#### Node Timing Model

The node timing model contains node-level timing information. This model corresponds to the tighter version of the offset-based response-time analysis [18] which is based on a transaction model with offsets developed by [19] and later on, extended by many researchers, e.g., [20, 21]. A node, $\Gamma$, consists of a set of $k$ transactions $\Gamma_1, \ldots, \Gamma_k$. Each transaction $\Gamma_i$ is activated by mutually independent events, i.e., the phasing between them is arbitrary. The activating events can be a periodic sequence of events with a period $T_i$. In case of sporadic

events, $T_i$ denotes the minimum inter-arrival time between two consecutive events.

There are $|\Gamma_i|$ tasks in a transaction $\Gamma_i$ and each task may not be activated until a certain time, called an *offset*, elapses after the arrival of the external event. By task activation we mean that the task is released for execution. A task is denoted by $\tau_{ij}$. The first subscript, *i*, specifies the transaction to which this task belongs and the second subscript, *j*, denotes the index of the task within the transaction.

A task, $\tau_{ij}$, is defined by the following attributes: a priority ($P_{ij}$), a worst-case execution time ($C_{ij}$), an offset ($O_{ij}$), maximum release jitter ($J_{ij}$), an optional deadline ($D_{ij}$), maximum blocking time which is the maximum time the task has to wait for a resource that is locked by a lower priority task ($B_{ij}$). In order to calculate the blocking time for a task, usually, a resource locking protocol like priority ceiling or immediate inheritance is used. Each task has a worst-case response time denoted by $R_{ij}$. In this model, there are no restrictions placed on offset, deadline or jitter, i.e., they can each be either smaller or greater than the period.

### Network Timing Model

This model contains network-level timing information of a DRE system. A network consists of a number of nodes that are connected through a real-time network. Currently, the model supports Controller Area Network (CAN) and its higher-level protocols such as CANopen, CAN for Military Land Systems domain (MilCAN), HCAN [22], etc. If a task on one node intends to communicate with a task on another node, it queues a message in the send queue of its node. The network communication protocol ensures the arbitration and transmission of all messages over the network.

Each message $m$ has the following attributes: a unique identifier ($ID_m$); transmission type showing whether the message is periodic or sporadic or mixed [22], i.e., both periodic and sporadic ($TRANSMISSION\_TYPE$); a unique priority ($P_m$); transmission time ($C_m$); release jitter ($J_m$) which is inherited from the response time of the task queueing the message; data payload ($s_m$) in the message; period ($T_m$) in case of periodic transmission or minimum inter-arrival time ($MINT_m$) which is the minimum time that should elapse between the transmission of any two sporadic messages in case of sporadic transmission; blocking time ($B_m$) which is the maximum amount of time a message can be blocked by the lower priority messages; and worst-case response time ($R_m$).

### 5.4.2 System Tracing Model

In DRE systems, the transactions are usually distributed over several nodes. Hence, there exist event chains that may be distributed over more than one node. An event chain consists of a number of tasks that are in a sequence and have one common triggering ancestor (e.g., clock, internal and external events, etc.). In a particular distributed transaction, a task on one node communicates via network messages with the task on another node belonging to the same transaction. When there are event chains in a DRE system, the end-to-end timing model should not only contain timing related information but also the tracing information of the event chains. By tracing information, we mean proper sequencing and linking information among all tasks inside the chain. The extraction of tracing information of event chains in a distributed real-time system is more complex compared to a single node real-time system. In [23], we highlighted the issues concerning tracing of event chains in component-based real-time systems in a single node (uniprocessor) as well as in a distributed system. We revisit this problem in DRE systems.

### 5.4.3 Problem: Tracing of Event Chains

Consider a DRE system modeled with RCM as shown in Figure 5.1. There are two nodes in the system with three SWCs in node A and four SWCs in node B. SWCs communicate with each other by using both inter-node and intra-node communication. The intra-node communication takes place via connectors whereas, the inter-node communication takes place via a real-time network to which the nodes are connected. One event chain (distributed transaction) that is activated by a clock consists of four Software Circuits, i.e., SWC1, SWC2, SWC4 and SWC5 and is identified with a solid-line arrow in Figure 5.1. In this transaction, a clock triggers SWC1 which in turn triggers SWC2. SWC2 then sends a signal to the network. This signal is transmitted over the network in a message (frame) and is received by the SWC4 at the receiver node. SWC4 processes it and sends it to SWC5. The elapsed time between the arrival of a triggering event at the input of the task corresponding to SWC1 and the response of the task corresponding to SWC5 is referred to as the holistic or end-to-end response time of the distributed transaction and is also identified in Figure 5.1. The second event chain that is activated by an external event consists of three Software Circuits, i.e., SWC3, SWC6 and SWC7. It is identified by a broken-line arrow in Figure 5.1.

There may not be direct triggering connections between any two neigh-

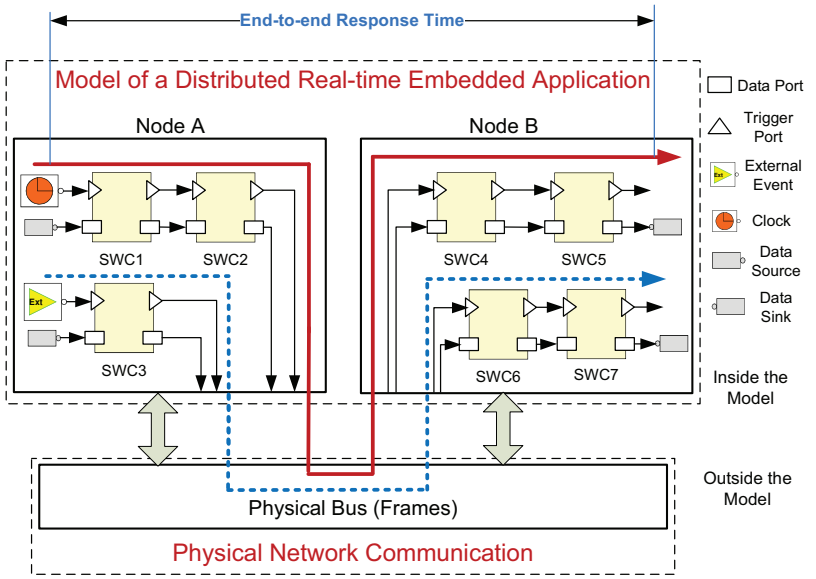


Figure 5.1: Event chains in distributed transactions

boring SWCs in the chain which is distributed over more than one node, e.g., SWC2 and SWC4 in Figure 5.1. In this case, SWC2 communicates with SWC4 by sending signals via the network. Here, the problem is that when a trigger signal is produced by SWC2, it may not be sent straightaway as a message on the network. A message may combine several signals and hence, there may be some waiting time for the signal to be sent on the network. The message may be sent periodically or sporadically or by any other rule defined by the underlying network protocol. When such event chains are modeled with a component model, it is not straightforward to trace them to extract the end-to-end timing model. For example, if a message is received at node B then the following information should be available to correctly link the received message in a chain: the $ID$ of the sender node; the $ID$ of the task that generated this message; the $ID$ of the destination node; and the $ID$(s) of the task(s) that should receive this message. In order to get a bounded end-to-end delay, a more important question is when and who will trigger the destination SWC when a message is received at node B.

**Discussion**

The existing modeling components in RCM do not provide enough support to trace and extract the corresponding timing information of event chains that are distributed over more than one node. Therefore, there is a need to introduce special modeling objects in the component model to provide the tracing information of event chains to extract end-to-end timing information for the timing model. Further, there is a need to model mapping between signals and messages and vice versa. SWCs inside a node communicate via signals whereas they communicate via messages if located on different nodes in a distributed transaction. Moreover, there is a need to model exit and entry points for RCM models. An exit point is where a message (data) leaves the model and is transmitted according to the protocol-specific rules of the network. Similarly, an entry point is where a message enters the model from the network. The reason for the need of modeling exit and entry points for RCM models is to get the bounded delays for extra-model medium. We propose the extraction of this information into a tracing model.

We believe that the issues discussed in this section may occur during the development of any other component model for distributed real-time systems that uses a pipe-and-filter communication mechanism for component interconnection, e.g., ProCom Component Model[12], COMDES [24], etc. The problem of tracing event chains may also exist in any type of "inter-model signaling", where a signal leaves one model (e.g., a node, or a core, or a process) and appears again in some other model. The requirement for the end-to-end timing analysis is that the "extra-model medium" can give bounded delays for the signal.

## 5.5 Extraction of End-to-end Timing Model

In this section, we resolve the issues discussed in the previous section. Moreover, we provide an example of a two-node DRE application modeled with RCM to show the applicability of our approach. Finally, we present the extraction of end-to-end timing model in Rubus-ICE.

### 5.5.1 Proposed Solution

#### Addition of Special Components in RCM

In order to model real-time network communication and legacy communication in DRE systems, we introduced special purpose Software Circuits in RCM, i.e., Output Software Circuit (OSWC) and Input Software Circuit (ISWC) in [8]. There is one OSWC for each message that a node sends to the network. Similarly, there is one ISWC for each message that a node receives from the network. We also introduced a new object in RCM, i.e., the Network Specification (NS) that represents the model of communication in a physical network [8]. There is one NS for each network protocol. NS contains Signal Mapping which includes the following information: How are signals mapped to messages? How many signals a message contains? How are signals encoded in a message at the sender node? How are signals decoded from a message at the receiving node? etc. The model representation of OSWC, ISWC and NS in a two-node DRE system is shown in Figure 5.2. The internal structure of nodes and the model of network communication is omitted for simplicity.

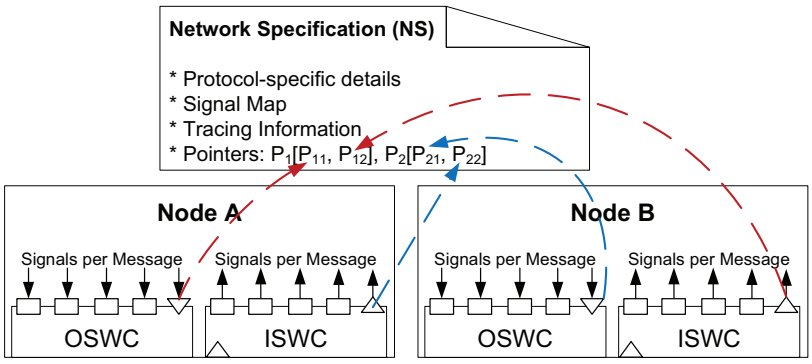


Figure 5.2: Model representation of OSWC, ISWC and NS in a two-node DRE system

#### Tracing Event Chains in DRE Systems

The tracing information of all event chains in the modeled DRE application is provided in the Network Specification. We assign pointers (references) to the input trigger ports of OSWCs and the output trigger ports of ISWCs along the

same distributed transaction. All such pointers for all the event chains in the system are specified in the NS. Consider again the example of a two-node DRE system shown in Figure. 5.2. Assume that there are four signals per message in each node. Further, assume that both nodes send messages to each other. There is a pointer array $P_1$ that references the trigger in-port of OSWC in Node A and trigger out-port of ISWC in node B. Similarly, a pointer array $P_2$ is stored in NS that points to the trigger in-port of OSWC in Node B and trigger out-port of ISWC in node A. In this way, the event chains in distributed transactions can be traced.

#### Exit and Entry Points for RCM Models

When the OSWC component is triggered, it executes the required functionality (e.g., mapping of signals to a message) and then the data (message) is transferred from the RCM model of a node to the network controller or another model of communication network. Therefore, OSWC also represents the model of an exit point for RCM models. Similarly, ISWC component also represents the model of an entry point for RCM models. Since the trigger in-ports of all OSWC components and trigger out-ports of all ISWC components along a distributed transaction are referenced in NS, the end-to-end timing delay can be bounded by specifying the delay in extra-model medium.

### 5.5.2 Example DRE System Modeled with RCM

An example of a two-node DRE system modeled in RCM is shown in Figure 5.3. There are four SWCs in Node A while three SWCs in Node B. For inter-node communication, CAN or any high-level protocol of CAN (e.g., CANopen, HCAN, MilCAN, etc.) can be used. In this example, the nodes are connected to a CAN network. There are three event chains in the system that are distributed over both the nodes:

- $EC_1 : SWC1 \rightarrow SWC2 \rightarrow OSWC\_A1 \rightarrow ISWC\_B1 \rightarrow SWC5 \rightarrow SWC6$.
- $EC_2 : SWC3 \rightarrow OSWC\_A1 \rightarrow ISWC\_B1 \rightarrow SWC5 \rightarrow SWC6$.
- $EC_3 : SWC7 \rightarrow OSWC\_B1 \rightarrow ISWC\_A1 \rightarrow SWC4$.

The event chains $EC_1$ and $EC_2$ are triggered by an external event whereas the event chain $EC_3$ is triggered by a clock. The references to the trigger

ports of OSWC and ISWC in each event chain are specified in NS. The grey boxes outside the model are specific for each network communication protocol. In this example the grey boxes represent CAN SEND and CAN RECEIVE routines. The CAN SEND grey box represents a CAN controller in a node and is responsible for receiving messages from the corresponding OSWC and queueing them for transmission over the network.

When a message arrives at the receiving node, it is transferred by the physical network drivers to the CAN RECEIVE grey box which is responsible for raising an interrupt request and passing the message to the corresponding ISWC component. Upon receiving a message, an ISWC component decodes it, extracts signals from it, places the data on the corresponding data port (connected to the data in-port of the destination SWC) and triggers the corresponding trigger port by using the tracing information in NS. It should be noted that there can be more than one ISWC and OSWC components in a node. It can be seen from Figure 5.3 that OSWC and ISWC essentially make the exit and entry points for the node models.

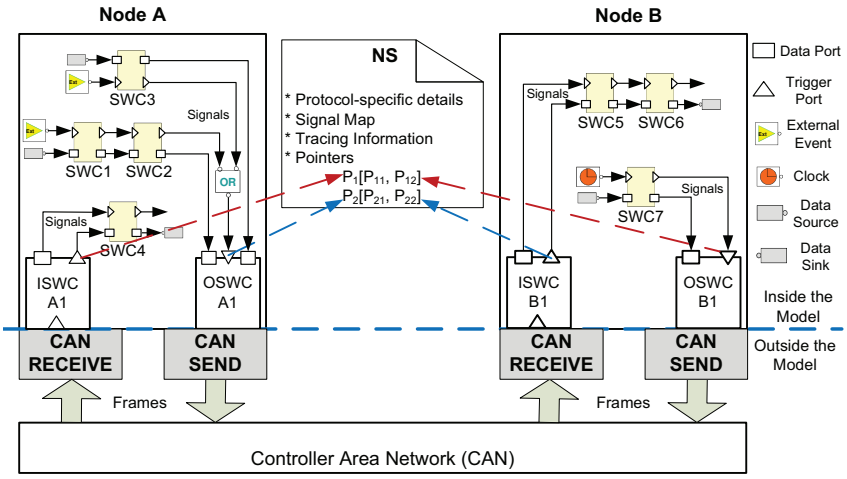


Figure 5.3: Example DRE system modeled with RCM

### 5.5.3 Extraction of End-to-end Timing Model in Rubus-ICE

In Rubus-ICE, a DRE application is modeled in Rubus Designer. It is then compiled to the Intermediate Compiled Component Model (ICCM). Apart from the compiled component model, ICCM file also includes timing and tracing information of the modeled system. The end-to-end timing model that is implemented in the Rubus Analysis Framework, extracts the required timing and tracing information from ICCM file as shown in Figure. 5.4. The end-to-end timing model consists of three models, i.e., node timing model, network timing model and system tracing model. From the extracted timing model, the Rubus Analysis Framework performs the end-to-end timing analysis and then provides the results, i.e., response times of individual tasks, response times of network messages, end-to-end response times of event chains, network utilization, etc., back to Rubus-ICE.

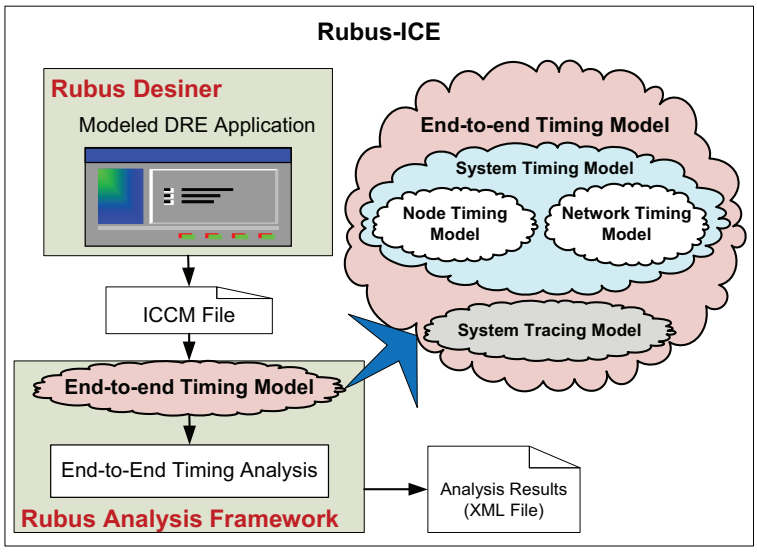


Figure 5.4: Extraction of end-to-end timing model in Rubus tool-suite

## 5.6 Conclusion and Future Work

In this paper, we discussed the extraction of the end-to-end timing model from component-based distributed real-time embedded (DRE) systems modeled with the industrially available component model, i.e., the Rubus Component Model (RCM). The purpose of extracting an end-to-end timing model is to perform the end-to-end timing analysis early during the development of DRE systems. We discussed and resolved various issues during the model extraction such as, extraction of timing information from all nodes and networks in the system, extraction of tracing model containing the tracing information of event chains in all distributed transactions, and modeling of exit and entry points for RCM models to provide timing bounds for extra-model medium. We also described the implementation of end-to-end timing model in the Rubus Analysis Framework.

Although, we discussed the extraction of end-to-end timing model for RCM, we believe it is also suitable for other component-models for the development of DRE systems that use a pipe-and-filter style for component interconnection. Moreover, our approach can be used for any type of "inter-model signaling", where a signal leaves one model (e.g. a node, or a core, or a process) and appears again in some other model. The requirement for end-to-end timing analysis is that the "extra-model medium" can give bounded delays for the signal.

In future, we plan to validate our methodology by making an industrial case study. Another interesting future work will be facilitating the exchange of analysis models and tools between RCM and other component models.

## Acknowledgement

This work is supported by Swedish Knowledge Foundation (KKS) within the project EEMDEF, the Swedish Research Council (VR) within project TiPCES, and the Strategic Research Foundation (SSF) with the centre PROGRESS. The authors would like to thank the industrial partners Arcticus Systems and BAE Systems Hägglunds for the cooperation.

# Bibliography


[1] Ivica Crnkovic and Magnus Larsson. *Building Reliable Component-Based Software Systems*. Artech House, Inc., Norwood, MA, USA, 2002.

[2] Thomas A. Henzinger and Joseph Sifakis. The Embedded Systems Design Challenge. In *Proceedings of the 14th International Symposium on Formal Methods (FM), Lecture Notes in Computer Science*, pages 1–15. Springer, 2006.

[3] TIMMO-2-USE. http://www.timmo-2-use.org/.

[4] TADL: Timing Augmented Description Language, Version 2. *TIMMO (TIMing MOdel), Deliverable 6*, October 2009. The TIMMO Consortium.

[5] TIMMO Methodology , Version 2. *TIMMO (TIMing MOdel), Deliverable 7*, October 2009. The TIMMO Consortium.

[6] Arcticus Systems. http://www.arcticus-systems.com.

[7] K. Hänninen et.al. The Rubus Component Model for Resource Constrained Real-Time Systems. In *3rd IEEE International Symposium on Industrial Embedded Systems*, June 2008.

[8] Saad Mubeen, Jukka Mäki-Turja, Mikael Sjödin, and Jan Carlson. Analyzable Modeling of Legacy Communication in Component-Based Distributed Embedded Systems. In *37th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), 2011*, pages 229 –238, Sep. 2011.

[9] AUTOSAR Techincal Overview, Version 2.2.2. AUTOSAR – AUTomotive Open System ARchitecture, Release 3.1, The AUTOSAR Consortium, Aug., 2008. http://autosar.org.

[10] Harald Heinecke et al. AUTOSAR – Current results and preparations for exploitation. In *Proceedings of the 7th Euroforum Conference*, EUROFORUM '06, May 2006.

[11] K. Richter et.al. A Timing Verification Methodology for AUTOSAR Series Development. In *3rd AUTOSAR Open Conference 2011*. http://www.autosar.org/download/conferencedocs11/15_AUTOSAR_AR-Methodik_Symtavision_final.pdf.

[12] Sverine Sentilles, Aneta Vulgarakis, Tomas Bures, Jan Carlson, and Ivica Crnkovic. A Component Model for Control-Intensive Distributed Embedded Systems. In *Proceedings of the 11th International Symposium on Component Based Software Engineering (CBSE2008)*, pages 310–317. Springer Berlin, October 2008.

[13] O. Scheickl and M. Rudorfer. Automotive Real Time Development Using a Timing-augmented AUTOSAR Specification. In *ERTS 2008*.

[14] N. Feiertag, K. Richter, J. Nordlander, and J. Jonsson. A Compositional Framework for End-to-End Path Delay Calculation of Automotive Systems under Different Path Semantics. In *Compositional Theory and Technology for Real-Time Embedded Systems, 2008. CRTS 2008. Workshop on*, dec. 2008.

[15] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Exploring Options for Modeling of Real-Time Network Communication in an Industrial Component Model for Distributed Embedded Systems. In *The 6th International Conference on Embedded and Multimedia Computing (EMC-2011)*, volume 102 of *Lecture Notes in Electrical Engineering*, pages 441–458. Springer Berlin / Heidelberg, 2011.

[16] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Modeling of legacy communication in distributed embedded systems. In *2nd Workshop on Model Based Engineering for Embedded Systems Design (M-BED 2011), located at Design, Automation & Test in Europe (DATE) Conference, 2011*, pages 1–6, March 2011.

[17] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Tracing event chains for holistic response-time analysis of component-based distributed real-time systems. In *23rd Euromicro Conference on Real-Time Systems (ECRTS 2011), WIP Session*. ACM SIGBED Review, July 2011.

[18] Jukka Mäki-Turja, , and Mikael Nolin. Tighter response-times for tasks with offsets. In *Real-time and Embedded Computing Systems and Applications Conference (RTCSA)*. Springer-Verlag, August 2004.

[19] Ken Tindell. Adding Time-Offsets to Schedulability Analysis. Technical report, Department of Computer Science, University of York, England, January 1994.

[20] J.C. Palencia and M. Gonzalez Harbour. Schedulability Analysis for Tasks with Static and Dynamic Offsets. *Real-Time Systems Symposium, IEEE International*, page 26, 1998.

[21] Jukka Mäki-Turja and Mikael Nolin. Efficient implementation of tight response-times for tasks with offsets. *Real-Time Syst.*, 40(1):77–116, 2008.

[22] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Extending schedulability analysis of controller area network (CAN) for mixed (periodic/sporadic) messages. In *Emerging Technologies Factory Automation (ETFA), IEEE 16th Conference on*, sept. 2011.

[23] Mubeen, Saad and Mäki-Turja, Jukka and Sjödin, Mikael. Tracing event chains for holistic response-time analysis of component-based distributed real-time systems. *SIGBED Review*, 8:48–51, September 2011.

[24] Xu Ke, K. Sierszecki, and C. Angelov. COMDES-II: A Component-Based Framework for Generative Development of Distributed Real-Time Control Systems. In *Embedded and Real-Time Computing Systems and Applications, RTCSA 2007. 13th IEEE International Conference on*, pages 199 –208, August 2007.

# Chapter 6

# Paper C: Extending Schedulability Analysis of Controller Area Network (CAN) for Mixed (Periodic/Sporadic) Messages

Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin.
In proceedings of the $16^{th}$ IEEE Conference on Emerging Technologies and Factory Automation (ETFA), pages 1-10, Toulouse, France, September, 2011.

**Abstract**

The schedulability analysis of Controller Area Network (CAN) developed by the research community is able to compute the response times of CAN messages that are queued for transmission periodically or sporadically. However, there are a few high-level protocols for CAN such as CANopen and Hägglunds Controller Area Network (HCAN) that support the transmission of mixed messages as well. A mixed message can be queued for transmission both periodically and sporadically. Thus, it does not exhibit a periodic activation pattern. The existing analysis of CAN does not support the analysis of mixed messages. We extend the existing analysis to compute the response times of mixed messages. The extended analysis is generally applicable to any high level protocol for CAN that uses any combination of periodic, event and mixed (periodic/event) transmission of messages.

## 6.1 Introduction

Often, real-time systems are employed in distributed systems. In such systems, also known as distributed real-time systems, the nodes (processors) communicate with each other by sending and receiving messages over a real-time network or a bus. Controller Area Network (CAN) [1, 2] is a real-time, event-triggered, serial communication bus protocol. It supports bus speeds of up to 1 mega bits per second. CAN is a largely used real-time network in automotive domain. Moreover, it finds its application in other domains such as, medical equipments, industrial control, etc. There are many high level protocols and commercial extensions of CAN developed for many industrial applications. These include CAN Application Layer (CAL) [3], CANopen [4], Hägglunds Controller Area Network (HCAN) [5], CAN for Military Land Systems domain (MilCAN) [6], DeviceNet, etc.

System providers of hard real-time systems are required to ensure that the system meets its deadlines. Moreover, the need for safety criticality in most of the hard real-time systems requires an evidence that the actions by the system will be provided in a timely manner (e.g. each action will be taken at a time that is appropriate to the environment of the system). Therefore, it is important to predict the timing behavior of such systems. In order to provide the evidence that each action in the system will meet its deadline, *a priori* analysis techniques, also known as schedulability analysis techniques, have been developed by the research community.

Response-Time Analysis (RTA) [7, 8] is a powerful, mature and well established schedulability analysis technique. It is a method to calculate upper bounds on the response times of tasks or messages in a real-time system or a real-time network respectively. In crux, RTA is used to perform a schedulability test which means it checks whether or not tasks (or messages) in the system (or network) will satisfy their deadlines. RTA applies to systems (or networks) where tasks (or messages) are scheduled with respect to their priorities and which is the predominant scheduling technique used in real-time operating systems (or real-time network protocols e.g., CAN) today [9].

Tindell et al. [10] developed schedulability analysis of CAN which was recognized by the automotive industry. Later on, the analysis was revisited and revised by Davis et al. [11]. The model of communication used by this analysis assumes that the messages are queued for transmission by the application tasks which are activated periodically or sporadically. However, there are a few high-level protocols and commercial extensions of CAN such as CANopen and HCAN, that support the transmission of mixed messages as well. A mixed

message contains both periodic and event signals. Thus a mixed message can be queued for transmission periodically as well as sporadically at the arrival of event signals. The current schedulability analysis of CAN does not support mixed messages.

In this paper, we extend the existing schedulability analysis of CAN to support the analysis of mixed messages. The extended analysis is able to find out the response times of periodic, event and mixed (periodic/event) CAN messages. The extended analysis is applicable to any high level protocol for CAN that uses any combination of periodic, event and mixed (periodic/event) transmission of messages. The motivation for this work comes from the activity of implementing the Holistic Response-Time Analysis (HRTA) [12] in the industrial tool suite, Rubus-ICE (Integrated Component development Environment) [13], that provides a component-based development environment for resource constrained distributed real-time systems.

The rest of the paper is organized as follows. In Section 6.2, we discuss the related work. In Section 6.3, we describe three different transmission patterns of CAN messages. In Section 6.4, we present the scheduling model for network communication. In Section 6.5, we visit the existing schedulability analysis of CAN and present the extended analysis. Finally, Section 6.6 concludes the paper.

## 6.2 Related Work

Liu and Layland [14] provided theoretical foundation for analysis of fixed-priority scheduled systems. Since then schedulability analysis of fixed-priority preemptive systems has been well developed. Joseph and Pandya published the first Response-Time Analysis (RTA) [15] for the simple task model presented by Liu and Layland which assumes independent periodic tasks.

There are many protocols such as CAN, TDMA (Time Division Multiple Access), TTCAN (Time-Triggered CAN), FlexRay, etc., that are used for real-time communication in distributed real-time systems. Schedulability analysis of these protocols has been developed by the research community. In this paper, we will focus only on the CAN protocol. Tindell et al. [10] developed the schedulability analysis of CAN by adapting the theory of fixed priority preemptive scheduling for uniprocessor systems. This analysis has been implemented in the analysis tools that are used in the automotive industry [16, 17]. Moreover, this analysis has served as basis for many research projects. Later on, this analysis was revisited and revised in [11]. The communication model

used in this analysis supports the analysis of CAN messages that are queued for transmission periodically or sporadically. This analysis does not support the response-times computation of CAN messages that are queued for transmission both periodically and sporadically.

Tindell [12] developed the holistic schedulability analysis for distributed hard real-time systems. Holistic analysis combines both the schedulability analysis of nodes (uniprocessors) and the network. This analysis is able to analyze a distributed real-time system that employs CAN or a simple TDMA protocol. As discussed earlier, this analysis does not support the response-time computation of mixed type messages.

In [18], Pop et al. provide a holistic schedulability analysis of distributed embedded systems in which the tasks are both time- and event-triggered. The analysis is developed for ST/DYN protocol bus that uses static and dynamic phases for sending messages. Static phase is split into time slots and each node transmits in its own slot. The dynamic phase is shared by all nodes and the contention is resolved by message priorities. As compared to this approach, we use CAN protocol for network communication and the messages are queued by the tasks (that require remote transmission), on each node, periodically or sporadically or both periodically and sporadically.

## 6.3 Transmission Patterns of a CAN Message

When CAN is employed for network communication in a distributed real-time system, each node (processor) is equipped with a CAN interface that connects the node to the bus [19]. Application tasks in each node, that require remote transmission, are assumed to queue messages for transmission over CAN bus. The messages are actually transmitted according to the protocol specification of CAN. The classical scheduling analysis of CAN [10] assumes that the tasks queueing CAN messages are invoked either by periodic events with a period or sporadic events with a minimum inter-arrival time. However, there are few high level protocols and commercial extensions of CAN in which the task that queues the messages can be invoked periodically as well as sporadically and hence, does not exhibit periodic activation patterns.

Throughout this paper, we will use the terms message and frame interchangeably since we only consider messages that will fit into one frame (maximum 8 bytes). For the purpose of using simple notation, we will call a CAN frame as PERIODIC, EVENT or MIXED if it is queued by an application task that is invoked periodically, sporadically or both (periodically/sporadically) respec-

tively.

### 6.3.1 Periodic and Event Transmissions

If all the signals contained in a message are periodic then the transmission type of the message is periodic. Such a message will be queued for transmission at periodic intervals. On the other hand, if all the signals contained in a message are of event type then the message is said to have event transmission type. Such a message will be queued for transmission as soon as an event occurs that changes the value of one or more signals contained in the message provided a Minimum Update Time ($MUT$) between the queueing of two successive event messages has elapsed. Hence, the transmission of an event frame is constrained by $MUT$.

### 6.3.2 Mixed (Periodic/Event) Transmission

If a message can be queued periodically as well as at the arrival of an event then the transmission type of a message is called mixed (periodic/event) or simply mixed transmission. We identified two different methods of implementing mixed messages for CAN protocol.

#### Method 1: Implementation of a Mixed Message

The CANopen protocol [20] provides an example of the first implementation method of a MIXED message. A mixed message can be queued for transmission at an arrival of an event provided an Inhibit Time has expired. The Inhibit Time is the minimum time that must be allowed to elapse between the queueing of two consecutive messages. A mixed message can also be queued periodically at the expiry of an Event Timer. Hence, the expiry of an Event Timer is considered as an additional event for queueing of a mixed message. The Event Timer is reset every time the message is queued. It should be noted that once a mixed message is queued for transmission, any additional queueing of the same message will not take place during the Inhibit Time [20]. The transmission pattern of a mixed message in CANopen is illustrated in Figure 6.1. The down-pointing arrows (labeled with numbers) symbolize the queueing of messages while the upward lines (labeled with alphabets) represent arrival of the events.

In Figure 6.1, message 1 is queued for transmission as soon as an event $A$ arrives (assume that the Inhibit Timer was expired). In this case, the Event

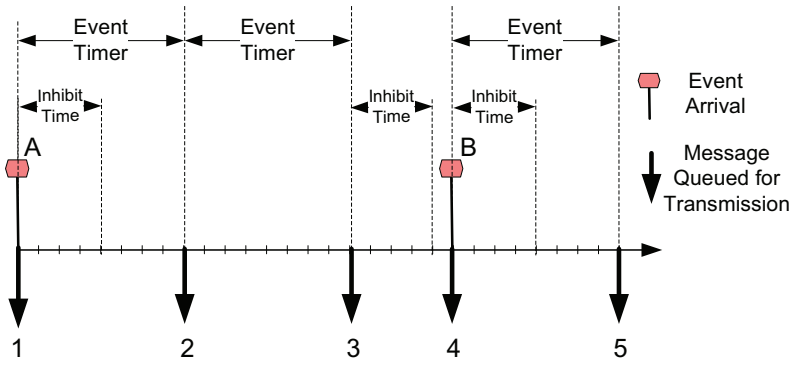


Figure 6.1: Transmission pattern of a Mixed Message in CANopen

Timer is reset along with the Inhibit Time. As soon as the Event Timer expires, message 2 is queued for transmission and both the Event Timer and Inhibit Time are reset. Similarly, message 3 is queued for transmission because of the expiry of the Event Timer. When an event $B$ arrives, message 4 is immediately queued for transmission because the Inhibit Time has already expired. Note that the Event Timer is also reset at the same time when the message 4 is queued. The message 5 is transmitted because of the expiry of the Event Timer. Hence, there exist a dependency relationship between the Inhibit Time and the Event Timer.

### Method 2: Implementation of a Mixed Message

The HCAN protocol [5] provides an example of the second implementation method of a MIXED message. A mixed message defined by HCAN protocol contains signals of which some are periodic and some are of event type. A mixed message is queued for transmission not only periodically but also, as soon as, an event occurs that changes the value of one or more event signals provided $MUT$ between the queueing of two successive event messages has elapsed. Hence, the transmission of a mixed message due to arrival of events is constrained by $MUT$. The transmission pattern of a mixed message is illustrated in Figure 6.2.

In Figure 6.2, message 1 is queued for transmission because of the partly periodic nature of a mixed message. As soon as the event $A$ arrives, message 2 is queued. When the event $B$ arrives it is not queued immediately because

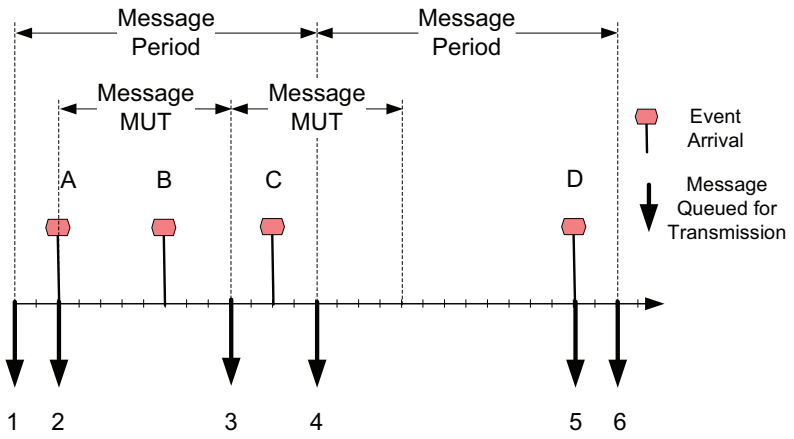


Figure 6.2: Transmission pattern of a Mixed Message in HCAN

$MUT$ is not expired yet. As soon as $MUT$ expires, message 3 is queued. Message 3 contains the signal changes that correspond to event $B$. Similarly, a message is not immediately queued when an event $C$ arrives because the $MUT$ is not expired. Message 4 is queued because of the periodicity. It should be noted that although, $MUT$ was not yet expired, the event signal corresponding to event $C$ was packed in message 4 and queued as part of the periodic message. Hence, there is no need to queue an additional event message when $MUT$ expires. It should be noted that the periodic transmission of a mixed message cannot be blocked by the event transmission. When an event $D$ arrives, an event message 5 is immediately queued because the $MUT$ has already expired. Message 6 is queued due to the periodicity.

### Discussion

In the first method, the Event Timer is reset every time a mixed message is queued for transmission. The most natural interpretation of a mixed message from the specification of CANopen is that there is an implicit requirement that the periodicity of transmission of a mixed message can never be higher than the Inhibit Time [4] [21]. Hence, it can be assumed that in the worst case, a mixed message is queued for transmission every time the Inhibit Timer expires. Therefore, the original CAN analysis can be used for mixed messages in the first method.

However, the second method of implementing a mixed message is more complex because the periodic transmission is independent of the event transmission. In other words, the Event Timer is not reset with every event transmission. In this case, for the purpose of analysis we need to treat a mixed message as two separate message streams with same IDs and priorities. This calls for the need of new analysis for mixed CAN messages. In addition, the existing analysis does not support any two messages with same IDs and equal priorities, which also requires extension of the original analysis.

## 6.4 Network Scheduling Model

In this section, we discuss the network scheduling model that will be used in the development of extended analysis for mixed type CAN messages. This model is an extension to the communication model that was developed by Tindell et al. [10] for the response-time analysis of Controller Area Network (CAN) messages. The existing model supports the scheduling of messages that are queued for transmission periodically (PERIODIC messages) or sporadically (EVENT messages). We will extend this model to support the analysis of messages that are queued periodically as well as sporadically (MIXED messages).

Each CAN message $m$ has an $ID_m$ which is a unique identifier. Associated to each message is a $FRAME\_TYPE$ that specifies whether the frame is a Standard or an Extended CAN frame. The difference between the two frame types is that a standard CAN frame uses an 11-bit identifier whereas an extended CAN frame uses a 29-bit identifier. There is a $TRANSMISSION\_TYPE$ of each message that specifies whether the message is PERIODIC or EVENT or MIXED (both PERIODIC and EVENT). Each message has a unique priority ($P_m$), transmission time ($C_m$) and queueing jitter ($J_m$) which is inherited from the response time of the task queueing the message.

Each message can carry a data payload that ranges from 0 to 8 bytes. This number is specified in a header field of the frame called Data Length Code (DLC) and denoted by $s_m$. In case of PERIODIC transmission, each frame has a period, denoted by $T_m$. In case of EVENT transmission, each frame has a $MUT_m$ that refers to the minimum time that should elapse between the transmission of any two EVENT frames. Each message has a blocking time $B_m$ which refers to the largest amount of time this message can be blocked by any lower priority message. Each message has a worst-case response time, denoted by $R_m$, and defined as the longest time between the queueing of the message

(on the sending node) and the delivery of the message to the destination buffer (on the destination node).

When a message has a MIXED transmission type, we duplicate the message in the analysis model. Hence, each MIXED message has two copies which are treated as separate messages. One copy is the PERIODIC message and the other is an EVENT message. All the attributes of these duplicates, including ID, priority, release jitter, transmission time and blocking time, are the same except that the PERIODIC copy inherits $T_m$ while the EVENT copy inherits $MUT_m$.

It is important to note that CAN identifier of each message is unique and it also corresponds to its priority. As discussed earlier that in case of a MIXED message, we duplicate the message and the duplicates have the same identifier and priority. The existing analysis model [19][10] does not support any two messages with equal priorities.

## 6.5 Extending CAN Schedulability Analysis

In this section, we extend the scheduling analysis of CAN that was originally developed by Tindell et al. [10] and later revised by Davis et al. [11]. The extended analysis will be able to compute the response times of mixed type messages as well.

### 6.5.1 Existing Analysis

First of all, we quickly revisit the existing algorithms that are used to compute the response-times of CAN messages. Then we extend these algorithms to support the analysis of mixed type messages.

According to the existing analysis, the worst-case response time of a CAN message is given by the following equation:

$$R_m = J_m + \omega_m + C_m \tag{6.1}$$

where $m$ is the message under analysis. $J_m$ denotes the queueing jitter of $m$ and is inherited from the worst-case response time of the task that queues this message (sending task). $\omega_m$ represents the worst-case queueing delay and is equal to the longest time that elapses between the instant a message $m$ is queued by the sending task in the priority-ordered send queue and the instant when the message starts its transmission. In other words, $\omega_m$ is the interference caused by other messages to $m$.

It is important to mention that CAN uses fixed-priority non-preemptive scheduling and therefore, a message cannot be interfered by higher priority messages during its transmission on the bus. Whenever we use the term interference, it refers to the amount of time the message has to wait in the send queue because the higher priority messages win the arbitration and hence, the right of transmission before the message under analysis.

$\omega_m$ is given by the following recursive equation:

$$\omega_m^{n+1} = B_m + \sum_{\forall k \in hp(m)} \left\lceil \frac{\omega_m^n + J_k + \tau_{bit}}{T_k} \right\rceil C_k \tag{6.2}$$

In (6.2), $hp(m)$ refers to the set of all messages in the system that have higher priority than $m$. $\tau_{bit}$ denotes the time required to transmit a single bit on CAN bus. Its value depends upon the speed of the bus. In order to solve the recursive equation given by (6.2), initial value of $\omega_m^n$ can be taken equal to the blocking time, $B_m$, as given by the following equation:

$$\omega_m^0 = B_m \tag{6.3}$$

$B_m$ represents the maximum time for which $m$ can be blocked by the lower priority messages. It is equal to the largest transmission time of any message in the set of all the lower priority messages compared to the priority of $m$ and is given by the following equation:

$$B_m = \max_{\forall k \in lp(m)} (C_k) \tag{6.4}$$

where, $lp(m)$ refers to the set of all messages in the system that have lower priority than message $m$.

In (6.2), $C_m$ is the transmission time of $m$. It represents the longest time it takes for $m$ to be transmitted over the bus. The transmission time of the message is computed according to [11] as given by the following equation:

$$C_m = \left( g + 8s_m + 13 + \left\lfloor \frac{g + 8s_m - 1}{4} \right\rfloor \right) \tau_{bit} \tag{6.5}$$

where $s_m$ is the Data Length Code. It refers to the number of data bytes in a CAN data message. It can have any integer value from 0 to 8. $g$ is equal to $34$ and $54$ for standard and extended CAN frame formats respectively. For a Standard CAN identifier, (6.5) can be simplified as follows.

$$C_m = (55 + 10s_m)\tau_{bit} \tag{6.6}$$

Similarly, the transmission time of $m$ for an Extended CAN identifier is given by the following equation.

$$C_m = (80 + 10s_m)\tau_{bit} \tag{6.7}$$

In [11], Davis et al. made an observation that it is possible in the case of fixed-priority non-preemptive scheduling that a higher priority task may be waiting for transmission when a message $m$ finishes its transmission. Hence, they proposed to analyze all the instances of $m$ that lie in the level-m busy period.

In order to calculate the worst-case response time of a CAN message, the number of instances of $m$ that become ready for transmission before the end of the busy period should be known first. Then the response time of each instance of $m$ should be computed. The largest value from the response time of all instances should be picked up as the worst-case response time of $m$. The length of a priority level-m busy period, $t_m$, is given by the following recursive equation:

$$t_m^{n+1} = B_m + \sum_{\forall k \in hep(m)} \left\lceil \frac{t_m^n + J_k}{T_k} \right\rceil C_k \tag{6.8}$$

where, $hep(m)$ refers to the set of all messages in the system that have equal or higher priority than $m$. In order to solve this recursive equation, initial value of $t_m^n$ can be taken equal to the transmission time of $m$, i.e.

$$t_m^0 = C_m \tag{6.9}$$

The right hand side of (6.8) is a monotonic non-decreasing function of $t_m$. The recursive equation (6.8) is guaranteed to converge if the bus utilization for messages of priority level $m$ and higher, denoted by $U_m$, is less than 1. $U_m$ is given by the following equation:

$$U_m = \sum_{\forall k \in hep(m)} \frac{C_k}{T_k} \tag{6.10}$$

thus,

$$U_m < 1 \tag{6.11}$$

The number of instances of $m$, denoted by $Q_m$, that becomes ready for transmission before the end of the busy period is given by the following equation:

$$Q_m = \left\lceil \frac{t_m + J_m}{T_m} \right\rceil \tag{6.12}$$

The response time of each instance of $m$ is calculated by the following equation:

$$R_m(q) = J_m + \omega_m(q) - qT_m + C_m \tag{6.13}$$

where $q$ is the message-instance number. The range of $q$ is shown below.

$$0 \leq q \leq Q_m - 1 \tag{6.14}$$

The queueing delay of each instant of the message $m$ is given by the following equation.

$$\omega_m^{n+1}(q) = B_m + qC_m + \sum_{\forall k \in hp(m)} \left\lceil \frac{\omega_m^n(q) + J_k + \tau_{bit}}{T_k} \right\rceil C_k \tag{6.15}$$

After the response time of all instances of the message $m$ have been computed, its worst-case response time can be found by selecting the largest value as given by the following equation.

$$R_m = max(R_m(q)), \qquad \forall\, 0 \leq q \leq (Q_m - 1) \tag{6.16}$$

### 6.5.2 Extended Analysis

In the extended schedulability analysis of CAN, we treat a message differently based on its transmission type. In order to keep the notations simple and consistent, we define a function $\xi(m)$ that represents the transmission type of a message $m$. It can be either periodic or event or mixed. Formally, the domain of this function can be defined as:

$$\xi(m) \in [\text{PERIODIC}, \quad \text{EVENT}, \quad \text{MIXED}]$$

We assume that there are multiple slots for sending and receiving messages in the CAN controllers. Usually each slot has a single buffer [11]. If the previous instance of a message is not sent before the next then the previous instance is overwritten by the next one. In case of multiple buffers per slot,

we assume that the FiFo (First in First out) policy is used to send the multiple instances of a message.

We discuss two cases. In the first, we assume that a message under analysis has a transmission type either periodic or event. Whereas in the second case, we consider that the message under analysis is of mixed transmission type.

**Case 1: When the Message Under Analysis is Periodic or Event**

When the transmission type of $m$ is PERIODIC or EVENT then the worst-case response time of each instance $q$ of this message is computed by the following equation:

$$R_m(q) = \begin{cases} J_m + \omega_m(q) - qT_m + C_m, & \text{if } \xi(\text{k}) = \text{PERIODIC} \\ J_m + \omega_m(q) - q(MUT_m) + C_m, & \text{if } \xi(\text{k}) = \text{EVENT} \end{cases} \tag{6.17}$$

This equation is similar to the response-time equation (6.13) in the existing analysis. In (6.17), $J_m$ represents the queueing jitter which is equal to the worst-case response time of the task that queues $m$. $C_m$ represents the transmission time of $m$. It is calculated according to the existing analysis using (6.6) or (6.7) depending upon the type of CAN frame identifier. If the transmission type of a message under analysis is PERIODIC then the message period is taken into account. However, if the transmission type of the message is EVENT, minimum update time is used in the above response-time equation.

The algorithms for the computation of the worst-case queueing delay ($\omega_m$) of $m$ should include the interference caused by all the other PERIODIC, EVENT and MIXED messages. The existing analysis accounts the interference caused by only PERIODIC and EVENT messages.

As we discussed in the communication model that when transmission type of a message is MIXED, we duplicate the message and designate the duplicates as a PERIODIC and EVENT copy of the MIXED message. It is important to note that all the attributes of the duplicates are the same as that of the original MIXED message except the PERIODIC copy inherits the period while the EVENT copy inherits minimum update time.

**Worst Case Queueing Delay of a Periodic or Event Message**

Each higher priority MIXED message should contribute more interference to the the message under analysis. The worst-case queueing delay, adapted from

(6.15) in the existing analysis, can be computed by the following recursive equation:

$$\omega_m^{n+1}(q) = B_m + qC_m + \sum_{\forall k \in hp(m)} I_k C_k \tag{6.18}$$

where $I_k$ is computed differently for different values of $\xi(k)$ (k is the index of any higher priority message) as shown below. Note that the interference by a higher priority MIXED message contains the contribution from both the duplicates.

$$I_k = \begin{cases} \left\lceil \frac{\omega_m^n(q)+J_k+\tau_{bit}}{T_k} \right\rceil, & \text{if } \xi(\text{k}) = \text{PERIODIC} \\ \left\lceil \frac{\omega_m^n(q)+J_k+\tau_{bit}}{MUT_k} \right\rceil, & \text{if } \xi(\text{k}) = \text{EVENT} \\ \left\lceil \frac{\omega_m^n(q)+J_k+\tau_{bit}}{T_k} \right\rceil + \left\lceil \frac{\omega_m^n(q)+J_k+\tau_{bit}}{MUT_k} \right\rceil, & \text{if } \xi(\text{k}) = \text{MIXED} \end{cases} \tag{6.19}$$

The initial value of $\omega_m^n$ can be taken equal to the blocking time of $m$ as given by (6.3). $B_m$ in (6.18) can be computed by the same method which is used in the existing analysis given by (6.4). This is because CAN uses fixed priority non-preemptive scheduling and any message can be blocked by only one message in the set of lower priority messages. Although we duplicate all the mixed messages, a message under analysis can only be blocked by either the periodic copy or the event copy of any lower priority MIXED message. It should be noted that both the copies of a MIXED message have the same transmission time, $C_m$. Hence $B_m$ is equal to the largest transmission time among all periodic, event and mixed messages in a set of lower priority messages with respect to the message under analysis.

**Length of the Busy Period**

The length of priority level-m busy period, denoted by $t_m$, is also adapted from the existing analysis as given in (6.8). It can be computed by the following recursive equation.

$$t_m^{n+1} = B_m + \sum_{\forall k \in hep(m)} I'_k C_k \tag{6.20}$$

where $I'_k$ is given by the following relation. Note that the contribution of both the duplicates of a MIXED message $k$ is taken into account, provided $k$ belongs to a set of equal or higher priority messages with respect to $m$.

$$I'_k = \begin{cases} \left\lceil \frac{t_m^n + J_k}{T_k} \right\rceil, & \text{if } \xi(k) = \text{PERIODIC} \\ \left\lceil \frac{t_m^n + J_k}{MUT_k} \right\rceil, & \text{if } \xi(k) = \text{EVENT} \\ \left\lceil \frac{t_m^n + J_k}{T_k} \right\rceil + \left\lceil \frac{t_m^n + J_k}{MUT_k} \right\rceil, & \text{if } \xi(k) = \text{MIXED} \end{cases} \tag{6.21}$$

In order to solve this recursive equation, $C_m$ can be used as an initial value of $t_m^n$ as shown in (6.9). The right hand side of (6.20) is a monotonic non-decreasing function of $t_m$. The recursive equation (6.20) is guaranteed to converge if the bus utilization for messages of priority level-m and higher, denoted by $U_m$, is less than 1. That is,

$$U_m < 1 \tag{6.22}$$

where $U_m$ is computed by the following equation:

$$U_m = \sum_{\forall k \in hep(m)} C_k I''_k \tag{6.23}$$

where $I''_k$ is given by the following relation:

$$I''_k = \begin{cases} \frac{1}{T_k}, & \text{if } \xi(k) = \text{PERIODIC} \\ \frac{1}{MUT_k}, & \text{if } \xi(k) = \text{EVENT} \\ \frac{1}{T_k} + \frac{1}{MUT_k}, & \text{if } \xi(k) = \text{MIXED} \end{cases} \tag{6.24}$$

In the above equation, the contribution by both the copies of all the mixed messages, lying in a set of equal and higher priority messages with respect to $m$, is clearly taken into account while calculating the bus utilization.

The number of instances of $m$, denoted by $Q_m$, that becomes ready for transmission before the busy period ends is given by the following equation (similar to the existing analysis):

$$Q_m = \begin{cases} \left\lceil \frac{t_m + J_m}{T_m} \right\rceil, & \text{if } \xi(\text{m}) = \text{PERIODIC} \\ \left\lceil \frac{t_m + J_m}{MUT_m} \right\rceil, & \text{if } \xi(\text{m}) = \text{EVENT} \end{cases} \tag{6.25}$$

The index of each message-instance is identified by $q$. The range of $q$ is shown as follows.

$$0 \leq q \leq Q_m - 1 \tag{6.26}$$

After computing the response time of all the instances of $m$, we select the largest value among these response times as the worst-case response time of $m$ as shown below.

$$R_m = max(R_m(q)), \qquad \forall\, 0 \leq q \leq (Q_m - 1) \tag{6.27}$$

**Case 2: When the Message Under Analysis is Mixed**

Since, a message with a MIXED transmission type is duplicated, we compute the response time of both the duplicates separately. For simplicity, we denote the PERIODIC and EVENT copies of a mixed message $m$ by $m_P$ and $m_E$ respectively. Let the worst-case response time of $m_P$ and $m_E$ be denoted by $R_{m_P}$ and $R_{m_E}$ respectively. The worst-case response time of $m$ is equal to the largest value between $R_{m_P}$ and $R_{m_E}$ as given by the following equation:

$$R_m = max(R_{m_P},\ \ R_{m_E}) \tag{6.28}$$

where, $R_{m_P}$ and $R_{m_E}$ are computed separately by adapting the existing analysis. Let us denote the total number of instances of messages $m_P$ and $m_E$, occurring in the priority level-m busy period, by $Q_{m_P}$ and $Q_{m_E}$ respectively. Assume that the index variable for message instances of $m_P$ and $m_E$ is denoted by $q_{m_P}$ and $q_{m_E}$ respectively. The range of $q_{m_P}$ and $q_{m_E}$ is shown by the following equations:

$$0 \leq q_{m_P} \leq (Q_{m_P} - 1) \tag{6.29}$$

similarly,

$$0 \leq q_{m_E} \leq (Q_{m_E} - 1) \tag{6.30}$$

The worst-case response time of $m_P$ is equal to the largest value among the response times of all its instances in the busy period as shown by the following equation.

$$R_{m_P} = max(R_{m_P}(q_{m_P})) \tag{6.31}$$

Similarly, the worst-case response time of $m_E$ is equal to the largest value among the response times of all its instances in the busy period. It is given by the following equation.

$$R_{m_E} = max(R_{m_E}(q_{m_E})) \tag{6.32}$$

The worst-case response time of each instance of $m_P$ and $m_E$ can be derived by adapting the equations for the computation of worst-case response time of PERIODIC and EVENT messages respectively, derived in case 1, as given by the following two equations:

$$R_{m_P}(q_{m_P}) = J_m + \omega_{m_P}(q_{m_P}) - q_{m_P}T_m + C_m \tag{6.33}$$

$$R_{m_E}(q_{m_E}) = J_m + \omega_{m_E}(q_{m_E}) - q_{m_E}MUT_m + C_m \tag{6.34}$$

The queueing jitter, $J_m$, is the same in both the equations (6.33) and (6.34). It is equal to the worst-case response time of the task that queues $m$. The transmission time, $C_m$, is also the same in these equations and is calculated according to the existing analysis by using (6.6) or (6.7) depending upon the type of CAN frame identifier. Although, both the duplicates of $m$ inherit same $J_m$ and $C_m$ from it, they experience different amount of worst-case queueing delay caused by other messages.

The worst-case queueing delay experienced by $m_P$ and $m_E$ is denoted by $\omega_{m_P}$ and $\omega_{m_E}$ in (6.33) and (6.34) respectively. $\omega_{m_P}$ and $\omega_{m_E}$ can be computed by adapting the algorithm for the computation of the worst-case queueing delay for PERIODIC and EVENT messages presented in (6.18). In this algorithm, we need to add the contribution of $m_P$ to the worst-case queueing delay experienced by $m_E$ and vice versa. It should be noted that the copies of a mixed message have equal priority and the existing analysis does not allow any two messages with equal priority.

**Effect of Self Interference in a Mixed Message**

In order to derive the contribution of one copy of a mixed message to the worst-case queueing delay of the other, consider three different cases, depicting the transmission pattern of a mixed message $m$, shown in Figure 6.3. In

the first case, we assume that $T_m$ is greater than $MUT_m$. This means that there could be more transmissions of the event copy compared to the periodic copy of $m$. Since the maximum update time between the queueing of any two event copies can be arbitrarily very long, it is also possible that there are fewer event transmissions than the periodic transmissions of $m$. In the second case, we assume that $T_m$ is equal to $MUT_m$. In this case, there could be equal transmissions of both the copies of $m$. In the third case, we assume that $T_m$ is smaller than $MUT_m$. This implies that the event transmissions will be less than the periodic transmissions of $m$.

It is important to note that in the example shown in Figure 6.3, there is a small offset between the first periodic and event transmission of $m$. This offset is used to maximize the queueing delay. If this offset is removed then only one frame will be queued corresponding to the first instance of both periodic and event copy. Moreover, the larger value between $T_m$ and $MUT_m$ is the integer multiple of the smaller in all the cases. This relationship along with the offset between $T_m$ and $MUT_m$ ensures that periodic and event transmission of $m$ will not overlap, there by, maximizing the queueing delay.

**Case (a): $\mathbf{T_m > MUT_m}$**

Let the message under analysis be $m_P$ and consider case (a) in Figure 6.3. An application task queues $m$ periodically with a period $T_m$ (e.g., equal to 9 time units). Moreover, the same task can also queue $m$ at the arrival of events (labeled with numbers 1-6). The queueing of $m_E$ is constrained by $MUT_m$ (e.g., equal to 3 time units). The first instance of $m_P$, i.e., ($q_{m_P} = 0$), is queued for transmission as shown by $m_P(0)$ in Figure 6.3. If event 1 had arrived at the same time as the queueing of $m_P(0)$ then the signals in $m_E(0)$ were updated as part of $m_P(0)$. In that case, $m_E(0)$ was not queued separately (this is the property of a mixed message). In order to maximize the contribution of $m_E$ on the queueing delay of $m_P$, $m_E(0)$ is queued just after the queueing of $m_P(0)$ as shown in all the cases in Figure 6.3. Therefore, $m_E(0)$ and subsequent instances of $m_E$ will have no contribution in the worst-case queueing delay of the first instance of $m_P$, i.e., $m_P(0)$.

Now, consider the second instance of $m_P$. All the instances of $m_E$ that are queued just before the queueing of $m_P(1)$ will contribute to its worst-case queueing delay. It can be observed in the case (a) that the first three instances of $m_E$ are queued before $m_P(1)$. Similarly, there are six instances of $m_E$ that are queued before $m_P(2)$.

Let $Q^P_{m_E}$ denotes the total number of instances of $m_E$ that are queued

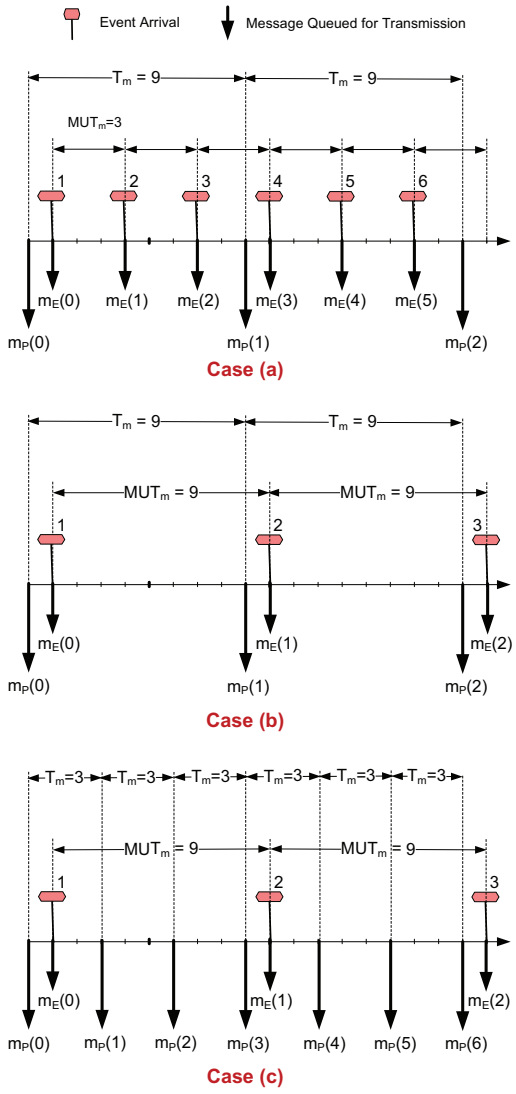


Figure 6.3: Demonstration of self interference in a MIXED message. Case (a) $\mathbf{T_m > MUT_m}$. Case (b) $\mathbf{T_m = MUT_m}$. Case (c) $\mathbf{T_m < MUT_m}$

before the $q^{th}_{m_P}$ instance of $m_P$. We can generalize $Q^P_{m_E}$ for the case (a) as follows:

$$Q^P_{m_E} = \left\lceil \frac{q_{m_P} T_m}{MUT_m} \right\rceil \tag{6.35}$$

for example, consider again the queueing of different instances of $m_E$ and $m_P$ in the case (a). Equation (6.35) yields the set $\{Q^P_{m_E} = 0, 3, 6, ...\}$ for the corresponding values in the set $\{q_{m_P} = 0, 1, 2, ...\}$. Thus the total number of instances of $m_E$ queued before each instance of $m_p$ computed by (6.35) are consistent with the case (a) in Figure 6.3.

**Case (b): $\mathbf{T_m = MUT_m}$**

Consider case (b) in which $T_m$ is equal to $MUT_m$. It can be observed from Figure 6.3 that there are 0, 1, and 2 instances of $m_E$ that are queued before $m_P(0)$, $m_P(1)$ and $m_P(2)$ respectively. When Equation (6.35) is used in case (b), we get the set $\{Q^P_{m_E} = 0, 1, 2, ...\}$ for the corresponding values in the set $\{q_{m_P} = 0, 1, 2, ...\}$. Therefore, (6.35) is also applicable on case (b).

**Case (c): $\mathbf{T_m < MUT_m}$**

Now, consider case (c) in which $T_m$ (equal to 3 time units) is smaller than $MUT_m$ (equal to 9 time units). The first instance of $m_E$, which is $m_E(0)$, will be queued before the queueing of $m_P(1)$, $m_P(2)$ and $m_P(3)$. Similarly, it can be seen from the figure that two instances of $m_E$, which are $m_E(0)$ and $m_E(1)$, will contribute to the worst-case queueing delay of $m_P(4)$, $m_P(5)$ and $m_P(6)$. (6.35) yields the set $\{Q^P_{m_E} = 0, 1, 1, 1, 2, 2, 2, ...\}$ for the corresponding values in the set $\{q_{m_P} = 0, 1, 2, 3, 4, 5, 6, ...\}$. Thus the total number of instances of $m_E$ queued before each instance of $m_p$ computed by Equation (6.35) are consistent with the case (c) in Figure 6.3.

Now we consider the effect of jitter on the instances of $m_E$ previous to $m_E(0)$ which can be queued just before $m_P(0)$ and hence, can contribute to the worst-case queueing delay of $m_P$. We assume a FiFo queue for the queueing of different instances of each message. By adding the jitter of $m_E$ to $Q^P_{m_E}$, equation (6.35) can be generalized for the three cases as follows.

$$Q^P_{m_E} = \left\lceil \frac{q_{m_P} T_m + J_m}{MUT_m} \right\rceil \tag{6.36}$$

The total number of instances of $m_P$ that are queued before the $q^{th}_{m_E}$ instance of $m_E$, denoted by $Q^E_{m_P}$, can be derived in a similar fashion. Thus

$Q^E_{m_P}$ can be computed by the following equation:

$$Q^E_{m_P} = \left\lceil \frac{q_{m_E} MUT_m + J_m}{T_m} \right\rceil \tag{6.37}$$

**Worst Case Queueing Delay of a Mixed Message**

The worst-case queueing delay of messages $m_P$ and $m_E$ can be computed by adapting (6.18) as follows.

$$\omega^{n+1}_{m_P}(q_{m_P}) = B_m + q_{m_P} C_m + \sum_{\forall k \in hp(m)} I_{k_P} C_k + Q^P_{m_E} C_m \tag{6.38}$$

$$\omega^{n+1}_{m_E}(q_{m_E}) = B_m + q_{m_E} C_m + \sum_{\forall k \in hp(m)} I_{k_E} C_k + Q^E_{m_P} C_m \tag{6.39}$$

Where, $I_{k_P}$ and $I_{k_E}$ are given by the following equations.

$$I_{k_P} = \begin{cases} \left\lceil \frac{\omega^n_{m_P}(q_{m_P}) + J_k + \tau_{bit}}{T_k} \right\rceil, & \text{if } \xi(k) = \text{PERIODIC} \\ \left\lceil \frac{\omega^n_{m_P}(q_{m_P}) + J_k + \tau_{bit}}{MUT_k} \right\rceil, & \text{if } \xi(k) = \text{EVENT} \\ \left\lceil \frac{\omega^n_{m_P}(q_{m_P}) + J_k + \tau_{bit}}{T_k} \right\rceil + & \\ \left\lceil \frac{\omega^n_{m_P}(q_{m_P}) + J_k + \tau_{bit}}{MUT_k} \right\rceil, & \text{if } \xi(k) = \text{MIXED} \end{cases} \tag{6.40}$$

$$I_{k_E} = \begin{cases} \left\lceil \frac{\omega^n_{m_E}(q_{m_E}) + J_k + \tau_{bit}}{T_k} \right\rceil, & \text{if } \xi(k) = \text{PERIODIC} \\ \left\lceil \frac{\omega^n_{m_E}(q_{m_E}) + J_k + \tau_{bit}}{MUT_k} \right\rceil, & \text{if } \xi(k) = \text{EVENT} \\ \left\lceil \frac{\omega^n_{m_E}(q_{m_E}) + J_k + \tau_{bit}}{T_k} \right\rceil + & \\ \left\lceil \frac{\omega^n_{m_E}(q_{m_E}) + J_k + \tau_{bit}}{MUT_k} \right\rceil, & \text{if } \xi(k) = \text{MIXED} \end{cases} \tag{6.41}$$

$B_m$ in equations 6.38 and 6.39 can be computed by the same method which is used in the existing analysis given by (6.4). By using the values of $Q^P_{m_E}$ and $Q^E_{m_P}$ from (6.36) and (6.37) in equations (6.38) and (6.39), we get:

$$\omega^{n+1}_{m_P}(q_{m_P}) = B_m + q_{m_P}C_m + \sum_{\forall k \in hp(m)} I_{k_P}C_k + \left\lceil \frac{q_{m_P}T_m + J_m}{MUT_m} \right\rceil C_m \tag{6.42}$$

$$\omega^{n+1}_{m_E}(q_{m_E}) = B_m + q_{m_E}C_m + \sum_{\forall k \in hp(m)} I_{k_E}C_k + \left\lceil \frac{q_{m_E}MUT_m + J_m}{T_m} \right\rceil C_m \tag{6.43}$$

In order to solve the recursive equations (6.40) and (6.41), initial values of $\omega^n_{m_P}(q_{m_P})$ and $\omega^n_{m_E}(q_{m_E})$ can be taken equal to the blocking time of the MIXED message $m$, i.e.

$$\omega^0_{m_P}(q_{m_P}) = \omega^0_{m_E}(q_{m_E}) = B_m \tag{6.44}$$

**Length of the Busy Period**

The length of priority level-m busy period, denoted by $t_m$, can be computed by using (6.20) that was developed for PERIODIC and EVENT messages. This is because (6.20) takes into account the effect of queueing delay from all the higher and equal priority messages. Since, the duplicates of a MIXED message inherit the same priority from it, the contribution of queueing delay from the duplicate is also covered in (6.20). Therefore, there is no need to compute $t_m$ for $m_P$ and $m_E$ separately. $t_m$ should be computed only once for a MIXED message $m$.

Although the length of the busy period is the same for $m_P$ and $m_E$, the number of instances of both the messages that become ready for transmission just before the end of busy period, i.e., $Q_{m_P}$ and $Q_{m_E}$ respectively, may be different. The reason is that the computation of $Q_{m_P}$ and $Q_{m_E}$ require $T_m$ and $MUT_m$ respectively and which may have different values. $Q_{m_P}$ and $Q_{m_E}$ can be computed by adapting (6.25) that was derived for the computation of the number of instances of PERIODIC and EVENT messages that become ready for transmission before end of the busy period. $Q_{m_P}$ and $Q_{m_E}$ are given by the following equations.

$$Q_{m_P} = \left\lceil \frac{t_m + J_m}{T_m} \right\rceil \tag{6.45}$$

$$Q_{m_E} = \left\lceil \frac{t_m + J_m}{MUT_m} \right\rceil \tag{6.46}$$

## 6.6 Conclusion

The schedulability analysis of Controller Area Network (CAN) developed by the research community can compute the response times of CAN messages that are queued by application tasks periodically or sporadically. The existing analysis does not support the analysis of mixed messages. A mixed message can be queued for transmission both periodically and sporadically. Mixed messages are used in some of the high-level protocols for CAN such as CANopen and HCAN. Hence, the context of this problem is very general and requires a new analysis to support mixed messages.

In this paper, we extended the existing schedulability analysis of CAN to support the analysis of mixed messages. The extended analysis is able to compute the response times of CAN messages with all types of transmission patterns, i.e., periodic, event and mixed. The extended analysis is applicable to any high level protocol or commercial extension of CAN that uses any combination of periodic, event and mixed (periodic/event) transmission of messages.

In future work, the extended analysis will be implemented in an existing industrial tool suite, the Rubus-ICE [22], that provides a complete component-based development environment for resource-constrained distributed real-time systems.

## Acknowledgement

This work is supported by Swedish Knowledge Foundation (KKS) within the project EEMDEF, the Swedish Research Council (VR) within project TiPCES, and the Strategic Research Foundation (SSF) with the centre PROGRESS. The authors would like to thank the industrial partners Arcticus Systems and BAE Systems Hägglunds for the cooperation.

# Bibliography


[1] Robert Bosch GmbH. CAN Specification Version 2.0. Postfach 30 02 40, D-70442 Stuttgart, 1991.

[2] ISO 11898-1. Road Vehicles interchange of digital information controller area network (CAN) for high-speed communication, ISO Standard-11898, Nov. 1993.

[3] CAL, CAN Application Layer for Industrial Applications, CiA Draft Standard DS-207, Version 1.1. *CAN-in-Automation*, Feb. 1996.

[4] CANopen high-level protocol for CAN-bus, Version 3.0. *NIKHEF, Amsterdam*, March 2000. http://www.nikhef.nl/pub/departments/ct/po/doc/CANopen.pdf.

[5] Jimmy Westerlund. Hägglunds Controller Area Network (HCAN), Network Implementation Specification. *BAE Systems Hägglunds, Sweden (internal document)*, April 2009.

[6] MilCAN (CAN for Military Land Systems domain). http://www.milcan.org/.

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

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

[9] Mikael Nolin, Jukka Mäki-Turja, and Kaj Hänninen. Achieving Industrial Strength Timing Predictions of Embedded System Behavior. In *ESA*, pages 173–178, 2008.

[10] K.W. Tindell, H. Hansson, and A.J. Wellings. Analysing real-time communications: controller area network (CAN). In *Real-Time Systems Symposium (RTSS) 1994*, pages 259 –263.

[11] Robert Davis, Alan Burns, Reinder Bril, and Johan Lukkien. Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems*, 35:239–272, 2007.

[12] Ken Tindell and John Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocess. Microprogram.*, 40:117–134, April 1994.

[13] Arcticus Systems. http://www.arcticus-systems.com.

[14] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *ACM*, 20(1):46–61, 1973.

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

[16] Volcano Network Architect (VNA). Mentor Graphics. http://www.mentor.com/products/vnd/communication-management/vna.

[17] L. Casparsson, A. Rajnak, K. Tindell, and P. Malmberg. Volcano - a revolution in on-board communications. In *Volvo Technology Report*, 1998.

[18] Traian Pop, Petru Eles, and Zebo Peng. Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems. In *Proceedings of the tenth international symposium on Hardware/software codesign*, CODES '02, pages 187–192, New York, NY, USA, 2002. ACM.

[19] Ken Tindell and Alan Burns. Guaranteeing Message Latencies on Controller Area Network (CAN). In *1st International CAN Conference, 1994*, pages 1 –11.

[20] CANopen Application Layer and Communication Profile. CiA Draft Standard 301. Version 4.02. February 13, 2002. www.ece.unh.edu/biolab/hof/public/CiA.

[21] Olaf Pfeiffer, Andrew Ayre, and Christian Keydel. *Embedded Networking with CAN and CANopen*. Annabooks, 2003.

[22] K. Hänninen et.al. Framework for real-time analysis in Rubus-ICE. In *Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on*, pages 782 –788, 2008.

# Chapter 7

# Paper D: Support for Holistic Response-time Analysis in an Industrial Tool Suite: Implementation Issues, Experiences and a Case Study

Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin.
Accepted for publication in proceedings of the $19^{th}$ IEEE Conference on Engineering of Computer Based Systems (ECBS), Novi Sad, Serbia, April, 2012.

**Abstract**

The process of implementing and integrating state-of-the-art real-time analysis techniques with an existing industrial tool suite for the development of Distributed Real-time Embedded (DRE) systems offers many challenges. The implementer has to not only code and implement the analysis in the tool suite, but also deal with several issues such as extraction of unambiguous timing and tracing information from the design model. In this paper we present an implementation of the Holistic Response-Time Analysis (HRTA) as a plug-in for an industrial tool suite Rubus-ICE that is used for component-based development of DRE systems. We discuss and solve the issues encountered and highlight the experiences gained during the process of implementation, integration and evaluation of HRTA plug-in. We also provide a proof of concept by modeling an automotive application (autonomous cruise control system) using component-based development and analyzing it with HRTA plug-in.

## 7.1 Introduction

In order to provide evidence that each action in the system will meet its deadline, *a priori* analysis techniques such as schedulability analysis have been developed by the research community. Response Time Analysis (RTA) [1, 2] is one of the methods to check the schedulability of a system. It calculates upper bounds on response times of tasks or messages in a real-time system or a network respectively. Holistic Response-Time Analysis (HRTA) [3, 4, 5] is an academic well established schedulability analysis technique to calculate upper bounds on the response times of event chains (distributed transactions) in a distributed real-time system. The process of transferring such academic research results to the tools for industrial use can be challenging.

A tool chain for the industrial development of component-based Distributed Real-time Embedded (DRE) systems consists of a number of tools such as designer, compiler, builder, debugger, simulator, etc. Often, a tool chain may comprise of tools that are developed by different tool vendors. The implementation of state-of-the-art complex real-time analysis techniques such as RTA and HRTA in such a tool chain is non-trivial because there are several issues that are encountered apart from merely coding and testing the analysis algorithms.

### 7.1.1 Goals and Paper Contributions

In this paper, we discuss the implementation of HRTA as a standalone plug-in in the industrial tool suite Rubus-ICE (Integrated Component development Environment) [6]. Our goals in this paper are as follows.

1. Transfer the state-of-the-art real-time analysis results, i.e., holistic response-time analysis to the tools for industrial use.

2. Discuss and solve several issues encountered during the implementation, integration and evaluation of HRTA as a plug-in for Rubus-ICE.

3. Discuss the experiences gained during the implementation, integration and evaluation of HRTA plug-in.

4. Provide a proof of concept by conducting an automotive-application case study.

We believe, the contributions in this paper may provide guidance for the implementation of complex real-time analysis techniques in any industrial tool suite

that supports a plug-in framework for the integration of new tools and allows component-based development of DRE systems.

### 7.1.2 Paper Layout

The rest of the paper is organized as follows. Section 7.2 presents the background and related work. Section 7.3 discusses the implemented analysis. Section 7.4 describes the experiences gained and issues encountered during the implementation of HRTA. Section 7.5 presents a test plan. In Section 7.6, we present an automotive case study by modeling and analyzing a DRE application. Section 7.7 concludes the paper and presents the future work.

## 7.2 Background and Related Work

### 7.2.1 The Rubus Concept

Rubus is a collection of methods and tools for model- and component-based development of dependable embedded real-time systems. Rubus is developed by Arcticus Systems [6] in close collaboration with several academic and industrial partners. Rubus is today mainly used for development of control functionality in vehicles. The Rubus concept is based around the Rubus Component Model (RCM) [7] and its development environment Rubus-ICE, which includes modeling tools, code generators, analysis tools and run-time infrastructure. The overall goal of Rubus is to be aggressively resource efficient and to provide means for developing predictable and analyzable control functions in resource-constrained embedded systems.

RCM expresses the infrastructure for software functions, i.e., the interaction between software functions in terms of data and control flow separately. The control flow is expressed by triggering objects such as internal periodic clocks, interrupts, internal and external events. In RCM, the basic component is called Software Circuit (SWC). The execution semantics of an SWC is simply: upon triggering, read data on data in-ports; execute the function; write data on data out-ports; and activate the output trigger. Recently, we extended RCM for the development of DRE systems by introducing new components [8, 9]. A detailed comparison of RCM with several component models is presented in [8].

Figure 7.1 depicts the sequence of main steps followed in Rubus-ICE from modeling of an application to the generation of code. The component-based

design of an application is modeled in the Rubus Designer tool. Then the compiler compiles the design model into an Intermediate Compiled Component Model (ICCM). After that the builder tool sequentially runs a set of plug-ins. Finally, a coder tool generates the code.

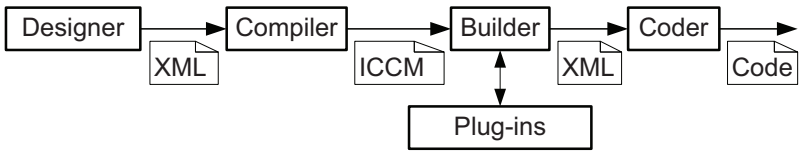


Figure 7.1: Sequence of steps from design to code generation in Rubus-ICE

### 7.2.2 Plug-in Framework in Rubus-ICE

The plug-in framework in Rubus-ICE [10] facilitates the implementation of state-of-the-art research results in an isolation (without needing Rubus tools) and their integration as add-on plug-ins (binaries or source code) with the integrated development environment. A plug-in is interfaced with a builder tool as shown in Figure 7.1. The plug-ins are executed sequentially which means that the next plug-in can execute only when the previous plug-in has run to completion. Hence, each plug-in reads required attributes as an input, runs to completion and finally writes the results to ICCM file. An Application Programming Interface (API) defines the services required and provided by a plug-in. Each plug-in specifies the supported system model, required inputs, provided outputs, error handling mechanisms and a user interface. Figure 7.2 shows a conceptual organization of a Rubus-ICE plug-in.

### 7.2.3 Response-Time Analysis

#### RTA of Tasks in a Node

Liu and Layland [11] provided theoretical foundation for analysis of fixed-priority scheduled systems. Joseph and Pandya published the first RTA [12] for the simple task model presented in [11]. Subsequently, it has been applied and extended in a number of ways by the research community. RTA is used to perform a schedulability test which means it checks whether or not tasks in the system will satisfy their deadlines. RTA applies to systems where

tasks are scheduled with respect to their priorities and which is the predominant scheduling technique used in real-time operating systems [13]. In [13], it is claimed that amongst the more traditional, analytical, schedulability analysis techniques, RTA of tasks with offsets stands out as the prime candidate because of its better precision and ability to analyze quite complex system behaviors.

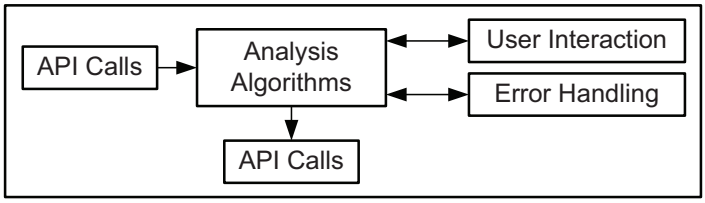


Figure 7.2: Conceptual organization of a plug-in in Rubus-ICE

Tindell [4] developed the schedulability analysis for tasks with offsets for fixed-priority systems. It was extended by Palencia and Gonzalez Harbour [5]. Later, Mäki-Turja and Nolin [14] reduced pessimism from RTA developed in [4, 5] and presented a tighter RTA for tasks with offsets by accurately modeling inter-task interference. We implemented RTA of tasks with offsets [14] as part of HRTA plug-in.

### RTA of Messages in a Network

There are many protocols such as, CAN (Controller Area Network), TTCAN (Time-Triggered CAN), FlexRay, etc., that are used in DRE systems. To stay focussed, we will consider only CAN and its high-level protocols. Tindell et al. [15] developed the schedulability analysis of CAN which has served as a basis for many research projects. Later on, this analysis was revisited and revised by Davis et al. [16]. The analysis in [15, 16] assumes that all CAN device drivers implement priority-based queues. In [17] Davis et al. pointed out that this assumption may become invalid when some nodes in a CAN network implement FIFO queues. Hence, they extended the analysis of CAN with FIFO queues as well. However, the existing analysis does not support mixed messages which are implemented by several high-level protocols for CAN. In [18, 19], Mubeen et al. extended the existing analysis to support RTA of mixed messages in the CAN network where some nodes use FIFO queues while others use priority queues.

**Holistic RTA**

It combines the analysis of nodes (uniprocessors) and a network. Hence, it computes the response times of event chains that are distributed over several nodes in a DRE system. In this paper, we consider a timing model that corresponds to the holistic schedulability analysis for DRE systems [3]. An example distributed transaction in a DRE system is shown in Figure 7.3. The holistic response time is equal to the elapsed time between the arrival of an event (corresponding to the brake pedal input) and the response time of Task4 (corresponding to the production of a signal for brake actuation). In [20], we discussed our preliminary findings about implementation issues that are encountered when HRTA is transferred to the industrial tools.

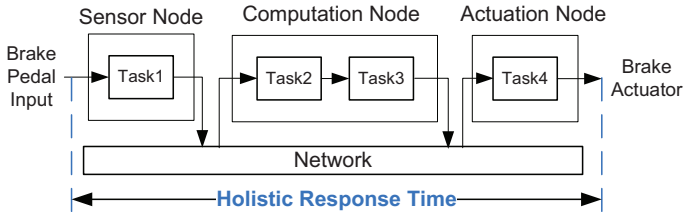


Figure 7.3: Holistic response-time in a distributed real-time system

### 7.2.4 Tools for Timing Analysis of DRE Systems

We briefly discuss few tool suits that provide similar real-time analysis support for DRE systems. The MAST tool suite [21] implements a number of state-of-the-art analysis algorithms for DRE systems. Among them is the offset-based analysis algorithm [4, 5] whose tighter version [14] is implemented in Rubus-ICE. The MAST model also allows visual modeling and analysis of real-time systems in a UML design environment. The Volcano Family [22] is a bunch of tools for designing, analyzing, testing and validating automotive embedded software systems. Among them, Volcano Network Architect (VNA) [23] is a communication design tool that supports the analysis of CAN and LIN networks. It also supports end-to-end timing analysis of a system with more than one networks. It implements RTA of CAN developed by Tindell et al. [15].

SymTA/S [24] is a tool for model-based timing analysis and optimization. It implements several real-time analysis techniques for single-node, multiprocessor and distributed systems. It supports RTA of software functions,

RTA of buss messages and end-to-end timing analysis of both single-rate and multi-rate systems. It is also integrated with the UML development environment to provide a timing analysis support for the applications modeled with UML [25]. RAPID RMA [26] implements several scheduling schemes and supports end-to-end analysis for single- and multiple-node real-time systems. It also allows real-time analysis support for the systems modeled with Real-Time CORBA [27].

The Rubus tool suite allows a developer to specify timing information and perform holistic response-time analysis at the modeling phase during component-based development of DRE systems. To the best of our knowledge, Rubus-ICE is the only tool suite that implements RTA of mixed messages in CAN [18] and a tighter version of offset-based RTA algorithm [14] as part of the holistic RTA.

## 7.3 Implemented Analysis in Rubus-ICE

We implemented HRTA as a standalone plug-in in Rubus-ICE. The plug-in can be used to compute the response times of individual tasks in a node, messages in a network and Distributed Transactions (DTs) in a distributed system.

### 7.3.1 Node Analysis

In order to analyze tasks in each node, we implemented RTA of tasks with offsets developed by [4, 5] and improved by [14].

### 7.3.2 Network Analysis

We implemented a network RTA that supports the analysis of CAN and its high-level protocols. It is based on the following RTA profiles for CAN.

1. RTA of CAN [15, 16].

2. RTA of CAN for mixed messages [18].

The above analysis assumes that the CAN nodes implement priority-ordered queues. The next step, as a future work, will be the implementation of CAN analysis that also supports FIFO ordered queues, i.e., RTA of CAN with FIFO queues [17] and RTA of CAN with FIFO Queues for Mixed Messages [19].

### 7.3.3 Holistic Analysis

The HRTA algorithm iteratively runs the analysis algorithms for node and network analysis. In the first step, release jitter of all messages and tasks in the system is assumed to be zero. The response times of all messages in the network and all tasks in each node are computed. In the second step attribute inheritance is carried out. This means that each message inherits a release jitter equal to the response time of its sender task (computed in the first step). Similarly, each receiver of a message inherits a release jitter equal to the response time of the message (computed in the first step). In the third step, response times of all messages and tasks are computed again. The newly computed response times are compared with the response times previously computed in the first step. The analysis terminates if the values are equal otherwise these steps are repeated. The conceptual view of HRTA that is implemented in Rubus-ICE is shown in Figure 7.4. The pseudocode of HRTA algorithm is shown in Algorithm 1.

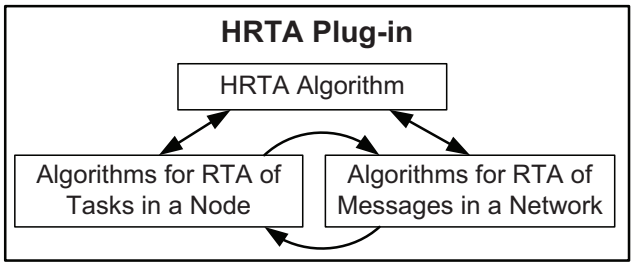


Figure 7.4: Conceptual view of HRTA plug-in in Rubus-ICE

## 7.4 Implementation Issues and Experiences

We discuss several issues encountered during the process of implementation and integration of HRTA as a standalone plug-in in Rubus-ICE. We also present our solution for each individual issue. Moreover, we discuss the summary of our experiences that are gained while translating theoretical results (HRTA) into the industrial tool suite.

**Algorithm 1** HRTA Algorithm

```
RT_Prev ← 0        // Initialize Response Time (RT) to zero
Repeat ← TRUE
while Repeat = TRUE do
  for all Messages and tasks in the system do
    Jitter_Msg ← RT_Sender
    Jitter_Receiver ← RT_Msg
    Compute RT of all messages
    Compute RT of all tasks in every node
    if RT > RT_Prev then
      RT_Prev ← RT
      Repeat ← TRUE
    else
      Repeat ← FALSE
    end if
  end for
end while
```

### 7.4.1 Extraction of Unambiguous Timing Information

One common assumption in HRTA is that the timing attributes required by the analysis are available as an input. However, when HRTA is implemented in a tool chain for the analysis of component-based DRE systems, the implementer has to not only code and implement the analysis, but also extract unambiguous timing information from the component model and map it to the inputs for the analysis model. This is because the design and analysis models are build upon different meta-models [28]. Often, the design model contains redundant timing information and hence, it is not trivial to extract unambiguous timing information for HRTA.

We divide the timing information (to be extracted) into two categories. The first category corresponds to the timing attributes of tasks (in each node) and network messages that are provided in the modeled application by the user. These timing attributes include Worst Case Execution Times (WCETs), periods, minimum update times, offsets, priorities, deadlines, blocking times, precedence relations in event chains, jitters, etc. In [9], we identified all the timing attributes of nodes, networks, transactions, tasks and messages that are required by HRTA. This timing information should be extracted from the modeled application and be made available as an input for HRTA.

The second category corresponds to the timing attributes that are not directly provided by the user but they must be extracted from the modeled application. For example, message period (in case of periodic transmission) or message inhibit time (in case of sporadic transmission) is often not specified by the user. These attributes must be extracted from the modeled application because they are required by the RTA of network communication. In fact, a message inherits the period or inhibit time from the task that queues this message. Thus, we assign a period or inhibit time to the message equal to the period or inhibit time of its sender task respectively.

However, the extraction of message timing attributes becomes complex when the sender task has both periodic and sporadic activation patterns. In such a case, not only the timing attributes of a message have to be extracted but also the transmission type of the message has to be identified. This issue can be visualized in an example shown in Figure 7.5. It should be noted that an Out Software Circuit (OSWC), shown in the figure, is one of the network interface components in RCM that sends a message to the network. The other network interface component is In Software Circuit (ISWC) that receives a message from the network [8].

In Figure 7.5(a), the sender task is activated by a clock and hence, the corresponding message is periodic. Similarly, the corresponding message is sporadic in Figure 7.5(b) because the sender task is activated by an event. However, the sender task in Figure 7.5(c) is triggered by both a clock and an event. Thus, the corresponding message will be a mixed message [18]. If there are periodic and sporadic messages in the modeled application, the HRTA plug-in uses the first profile of network analysis as discussed in the previous Section. On the other hand if the modeled application contains mixed messages as well, the second profile of network analysis is used. We extract the transmission type of a message from the modeled application as follows. If the sender of a message has a periodic or sporadic activation pattern then the message is assigned a periodic or sporadic transmission type respectively. However, if the sender is activated both periodically and sporadically, the message is assigned a mixed transmission type.

### 7.4.2 Extraction of Tracing Information from Distributed Transactions

In order to perform HRTA, correct tracing information of DTs should be extracted from the design model [29]. For this, we need to have a mapping among signals, data ports and messages. Consider the following DT in a two-node

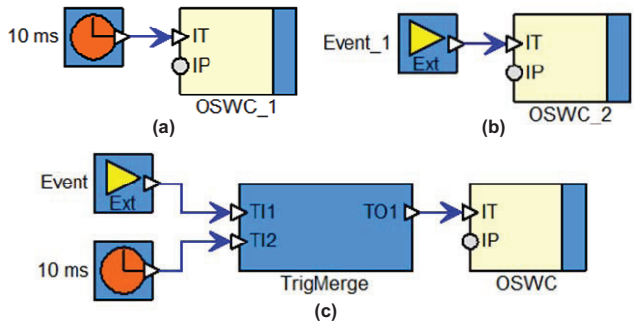


Figure 7.5: Extraction of transmission type of a message

DRE system modeled with RCM as shown in Figure 7.6.

$$SWC1 \rightarrow OSWC\_A \rightarrow ISWC\_B \rightarrow SWC2 \rightarrow SWC3$$

In this example, our focus is on the network interface components, i.e., OSWC and ISWC [8]. In order to compute the holistic response time of this DT, we need to extract tracing information from the component model. We identified a need for the following mappings in the component model.

- At the sender node, mapping between signals and input data ports of OSWC components.
- At the sender node, mapping between signals and a message that is sent to the network.
- At the receiver node, mapping between data output ports of ISWC components and the signals to be sent to the desired components.
- At the receiver node, mapping between message received from the network and the signals to be sent to the desired component.
- Mapping between multiple signals and a complex data port. For example, mapping of multiple signals extracted from a received message to a data port that sends a complex signal (structure of signals).
- Mapping of all trigger ports of network interface components along a DT as shown by a bidirectional arrow in Figure 7.6.

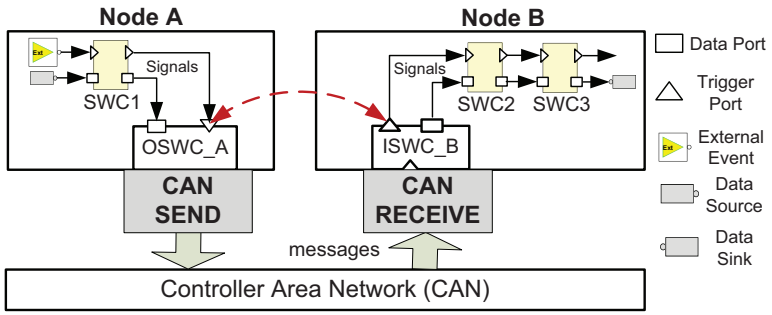


Figure 7.6: Two-node DRE system modeled in RCM

### 7.4.3 Impact of Design Decisions in Component Model on the Implementation of Analysis

Design decisions made in the component model can have indirect impact on the response times computed by the analysis. For example, design decisions could have impact on WCETs and blocking times which in turn have impact on the response times. In order to implement, integrate and test HRTA, the implementer needs to understand the design model (component model), analysis model and run-time translation of the design model. In the design model, the architecture of an application is described in terms of software components, their interconnections and software architectures. Whereas in the analysis model, the application is defined in terms of tasks, transactions, messages and timing parameters. At run-time, a task may correspond to a single component or chain of components. The run-time translation of a software component may differ among different component models.

### 7.4.4 Direct Cycles in Distributed Transactions

A direct cycle in a DT is formed when any two tasks located on different nodes send messages to each other. When there are direct cycles in a DT, the holistic analysis algorithm may run forever and may not produce converging results, i.e., the response times increase in every iteration.

Consider a two-node application modeled in RCM as shown in Figure 7.7 (a). The $OSWC\_A$ component in node A sends a message $m1$ to node B where it is received by $ISWC\_B$ component. Similarly, $OSWC\_B$ component in node B sends a message $m2$ to $ISWC\_A$ component in node A.

There are two options for run-time allocation of the network interface components (OSWC and ISWC) as shown in Figure 7.7 (b). First option is to allocate a network interface component to the task that corresponds to the immediate SWC, i.e., to the same task as that of the component that receives/sends the signals from/to it. Since $SWC\_A$ is immediately connected to both network interface components in node A, there will be only one task in node A denoted by $\tau_A$ as shown in Figure 7.7 (b). Similarly, $\tau_B$ is the run-time representation of $ISWC\_B$, $SWC\_B$ and $OSWC\_B$ components. It is obvious that the run-time allocation of network interface components in the first option results in direct cycles.

The direct cycles in DTs can be avoided by allocating each network interface component to a separate task as shown in the option 2 in Figure 7.7 (b). Although same messages are sent between the nodes, one task can not be both a sender and a receiver. No doubt there is a cycle between the nodes, but not a direct one. In this case, the holistic algorithm may produce converging response-time results and non-terminating execution of the plug-in may be avoided. It is interesting to note that the requirements and limitations of the analysis implementation provides feedback to the design decisions concerning the run-time allocation of modeling components.

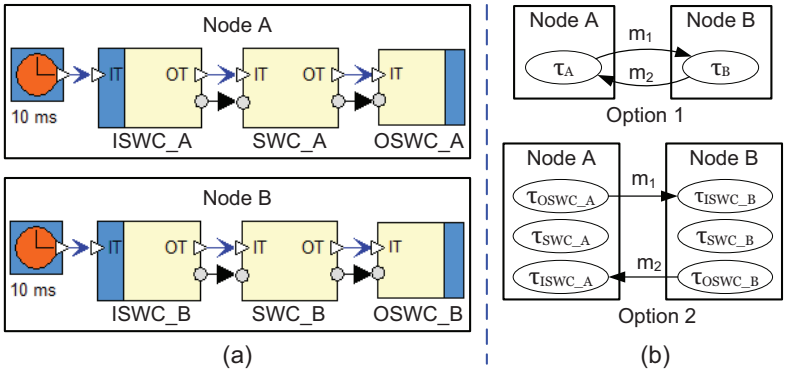


Figure 7.7: Run-time allocation of network interface components

### 7.4.5 Analysis of DRE Systems with Multiple Networks

In a DRE system, a node may be connected to more than one network. If a transaction is distributed over more than one network, the computation of its

holistic response time involves the analysis of more than one network. Consider an example of a DRE system with two networks, i.e., CAN and LIN as shown in Figure 7.8. There are five nodes in the system. Node 3 is a gateway node that is connected to both the networks. Consider a transaction in which $task1$ in $Node1$ sends a message to $task1$ in $Node5$ via $Node3$. The computation of holistic response time of this transaction will involve the computation of message response times in both CAN and LIN networks.

If a modeled system contains more than one network, we divide it into subsystems (each having a single network) and analyze them separately. In the above example, we first perform HRTA using CAN network. Then we provide the response times of the messages that are received at the gateway node as input jitters to the receiver tasks (attribute inheritance). Finally, HRTA of LIN network is performed. However, the implemented HRTA does not support the analysis of transactions that are distributed cyclically on multiple networks, i.e., the transactions that are distributed over more than one network and first and last task of the transaction are located on the same network.

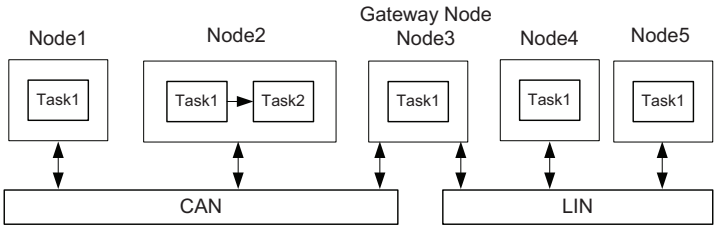


Figure 7.8: Multiple networks in a DRE system

### 7.4.6 Sequential Execution of Plug-ins in Rubus Plug-in Framework

The Rubus plug-in framework allows only sequential execution of plug-ins. Hence, a plug-in has to execute to completion and terminate before running the next plug-in. It should be noted that there exists a plug-in in Rubus-ICE that performs RTA of tasks in a node and it is already in the industrial use. There are two options to develop HRTA plug-in for Rubus-ICE, i.e., option A and B as shown in Figure 7.9.

The option A supports reusability by building the HRTA plug-in upon the existing Node RTA Plug-in. Thus, HRTA plug-in is built by integrating exist-

ing RTA plug-in and two new plug-ins, i.e., one implementing network RTA algorithms and the other implementing holistic RTA algorithm. In this case HRTA plug-in is very light weight. It iteratively uses the analysis results produced by the node and the network RTA plug-ins and accordingly provides new inputs to them until converging holistic response times are obtained. On the other hand, option B requires the development of HRTA plug-in from the scratch, i.e, implementing the algorithms of node, network and holistic RTA. This option does not support any reuse of existing plug-ins.

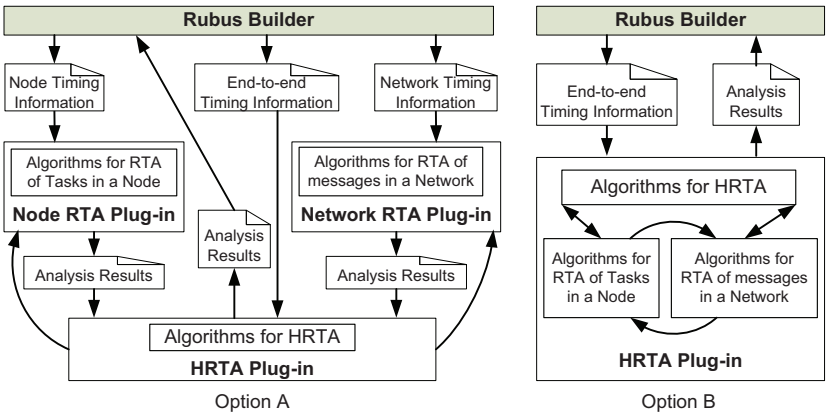


Figure 7.9: Options to develop HRTA Plug-in for Rubus-ICE

Since, option A allows the reuse of a pre-tested and heavy weight (having most complex algorithms compared to network and holistic RTA) node RTA plug-in, it is easy to implement and requires less time for implementation, integration and testing compared to option B. However, the implementation method in option A is not supported by the plug-in framework of Rubus-ICE because the plug-ins can only be sequentially executed and one plug-in can not execute the other. Hence, we had to select option B for the implementation of HRTA.

### 7.4.7 Presentation of Analysis Results

When HRTA of a modeled application has been performed, the next issue is how to present the analysis results. There can be a large number of tasks and messages in the system. It may not be appropriate to display the response time

of all the tasks and messages because it may contain a lot of useless information (if the user is not interested in all of it). Furthermore, presenting the response times of only DTs to the user may not be appropriate because there may be hundreds of DTs in a DRE application. A way around this problem is to provide the response times of only those tasks and DTs which have deadline requirements (specified by the user) or which produce control signals for external actuators (e.g., the analysis results of case study that will be discussed in Section 7.6). Apart from this, we also provide an option for the user to get detailed analysis results.

### 7.4.8 Interaction between the User and HRTA Plug-in

We identified that it is important to provide a progress report of HRTA plug-in during its execution. Based on the progress, the user should be able to interact with the plug-in while it is running. The HRTA algorithm iteratively runs the algorithms of node RTA and network RTA until converging values of the response times are computed or the computed response times exceed the deadlines (if deadlines are specified). We feel that it is important to display the number of iterations, running time and over all progress of the plug-in during its execution. Moreover, the user should be able to stop, rerun or exit the plug-in at any time.

### 7.4.9 Suggestions to Improve Schedulability Based on Analysis Results

If the analysis results indicate that the modeled system is unschedulable, it can be interesting if HRTA plug-in is able to provide suggestions (e.g., by varying system parameters) guiding the user to make the system schedulable. However, it is not trivial to provide such feedback because there can be so many reasons behind the system being not schedulable. The support for this type of feedback in HRTA plug-in will be provided in the future.

### 7.4.10 Requirement for Continuous Collaboration between Integrator and Implementer

Our experience of integrating HRTA plug-in with Rubus-ICE shows that there is a need of continuous collaboration between the integrator of the plug-in and its implementer especially in the phase of integration testing (see next Section). This collaboration is more obvious when the plug-in is developed in iso-

lation by the implementer (from research background) and integrated with the industrial tool chain by the integrator (with limited experience of integrating complex real-time analysis but aware of overall objective). A continuous consultation and communication was required between the integrator and the implementer for the verification of the plug-in. Examples of small DRE systems with varying architectures were created for the verification. The implementer had to verify these examples by hand. The integration testing and verification of HRTA plug-in was non-trivial and most tedious activity.

## 7.5 Testing and Evaluation

In this section we discuss our test plan for both standalone and integration testing of HRTA plug-in. Error handling and sanity checking routines make a significant part of the implementation. The purpose of these routines is to detect and isolate faults and present them to the user during the analysis. Our test plan contains the following sets of error handling routines.

- A set of routines evaluating the validity of all inputs: attributes of all nodes, transactions, tasks, networks and messages in the system.
- A set of routines evaluating the validity of linking and tracing information of all DTs in the system.
- A set of routines evaluating the validity of intermediate results that are iteratively inherited as inputs (e.g., a message inheriting the worst-case response time of the sender tasks as a release jitter).
- A set of routines evaluating the overload conditions during the analysis. For example, processor utilization exceeding 100%, presence of direct cycles in the system, etc. Since HRTA algorithm is iterative, the analysis may never terminate in the presence of these conditions.
- A set of routines evaluating variable overflow during the analysis.

### 7.5.1 Standalone Testing

Standalone testing means testing the implementation of HRTA before it is integrated as a plug-in with the Rubus builder tool. In other words, it refers to the testing of HRTA in an isolation. The following input methods were used for standalone testing.

1. Hard coded input test vectors.

2. Test vectors are read from external files.

3. Test vectors are generated using a test case generator (a separate program). This generator produces test cases with varying architectures. It also randomly inserts invalid inputs to check if the error handling routines are able to catch the errors.

The analysis results provided by the plug-in corresponding to the test vectors in the first two input methods were also verified by hand.

### 7.5.2 Integration Testing

Integration testing refers to the testing of HRTA plug-in after integrating it with the Rubus builder tool. Although standalone testing is already performed, the integration of HRTA with Rubus-ICE may induce unexpected errors. Our experience shows that integration testing is much more difficult and time consuming activity compared to standalone testing. The following input methods were used for integration testing.

1. Test vectors are read from external files.

2. Test vectors are manually written in ICCM file (see Figure 7.1) to make it appear as if test vectors were extracted from the modeled application.

3. Test vectors are automatically extracted from several DRE applications modeled with RCM.

The analysis results provided by the plug-in corresponding to all types of test cases were also verified by hand.

## 7.6 Automotive Case Study

We provide a proof of concept for the analysis approach that we implemented in Rubus-ICE by conducting an automotive case study. First, we model an Autonomous Cruise Control (ACC) system with RCM using Rubus-ICE. Then, we analyze the modeled ACC system using HRTA plug-in.

### 7.6.1 Autonomous Cruise Control System

A cruise control system is an automotive feature that allows a vehicle to automatically maintain a steady speed to the value that is preset by the driver. It uses velocity feedback from the speed sensor (e.g., a speedometer) and accordingly controls the engine throttle. However, it does not take into account traffic conditions around the vehicle. Whereas, an Autonomous Cruise Control (ACC) system allows the cruise control of the vehicle to adapt itself to the traffic environment without communicating with the surrounding vehicles. Often, it uses a radar to create a feedback of distance to and velocity of the preceding vehicle. Based on the feedback, it either reduces the vehicle speed to keep a safe distance and time gap from the preceding vehicle or accelerates the vehicle to match the preset speed specified by the driver [30].

An ACC system may be divided into four subsystems, i.e., Cruise Control, Engine Control, Brake Control and User Interface subsystem [31]. Figure 7.10 shows the block diagram of ACC system. The subsystems communicate with each other via a CAN network.

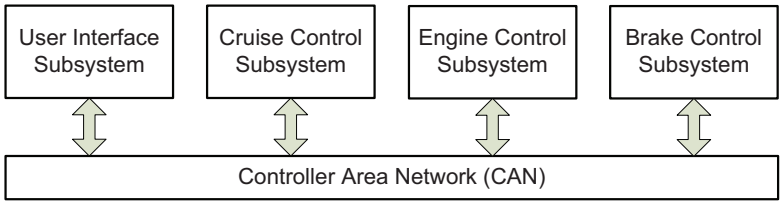


Figure 7.10: Block diagram of Autonomous Cruise Control System

#### User Interface Subsystem

The User Interface (UI) subsystem reads inputs (provided by the driver) and shows status messages and warnings on the display screen. The inputs are acquired by means of switches and buttons mounted on the steering wheel. These include Cruise Switch input (corresponding to ON/OFF/Standby states for ACC), Set Speed input (desired cruising speed set by the driver) and desired clearing distance from the preceding vehicle. Apart from user inputs, it also receives some other parameters from the rest of the subsystems via CAN network. These include linear and angular speed of the vehicle, i.e., kilometer per hour (KPH) and revolution per minute (RPM), status of manual brake

sensor, state of ACC subsystem, status messages and warnings to be displayed on the screen. Apart from showing status messages and warnings, it sends messages (including status of driver's input) to other subsystems.

### Cruise Control Subsystem

The Cruse Control (CC) subsystem receives user input information as a CAN message from the UI subsystem. From the received message it analyzes the state of the cruise control switch; if it is in ON state then it activates the cruise control functionality. It reads input from proximity sensor (e.g., radar) and processes it to determine the presence of a vehicle in front of it. Moreover, it processes the radar signals along with the information received from other subsystems such as vehicle speed to determine its distance from the preceding vehicle. Accordingly, it sends control information to the Brake Control and Engine Control subsystems to adjust the speed of the vehicle with the cruising speed or clearing distance from the preceding vehicle. It also receives the status of manual brake sensor from Brake Control subsystem. If brakes are pressed manually then the cruise control functionality is disabled. It also sends status messages to the UI subsystem.

### Engine Control Subsystem

The Engine Control (EC) subsystem is responsible for controlling the vehicle speed by adjusting engine throttle. It reads sensor input and accordingly determines engine torque. It receives CAN messages sent by other subsystems. The messages include information regarding vehicle speed, status of manual brake sensor, and input information processed by UI system. Based on the received information, it determines whether to increase or decrease engine throttle. It then sends new throttle position to the actuators that control engine throttle.

### Brake Control Subsystem

The Brake Control (BC) subsystem receives inputs from sensor for manual brakes status and linear and angular speed sensors connected to all wheels. It also receives a CAN message that includes control information processed by CC subsystem. Based on this feedback, it computes new vehicle speed. Accordingly, it produces control signals and sends them to the brake and back light actuators. It also sends CAN messages to other subsystems that carry information regarding status of manual brake, vehicle speed and RPM.

### 7.6.2 Modeling of ACC System with RCM in Rubus-ICE

In RCM, we model each subsystem as a separate node connected to a CAN network as shown in Figure 7.11. The selected speed of CAN bus is $500kbps$. The extended frame format is selected which means that all frames will use 29-bit identifier [32].

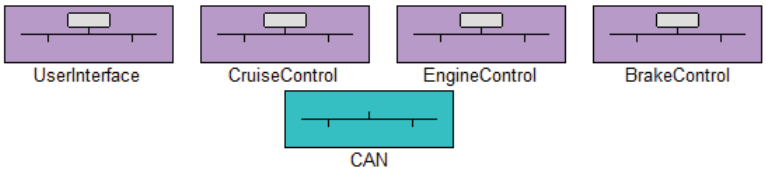


Figure 7.11: Model of Autonomous Cruise Control System in RCM

There are seven CAN messages that are sent by the nodes as shown in Figure 7.12. A signal data base that contains all the signals sent to the network is also shown. Each signal in the signal database is linked to one or more messages. The extracted attributes of all messages including data size ($s_m$), priority ($P_m$), transmission type ($\xi_m$) and period or minimum inter-arrival time ($T_m$) are listed in Table 1.

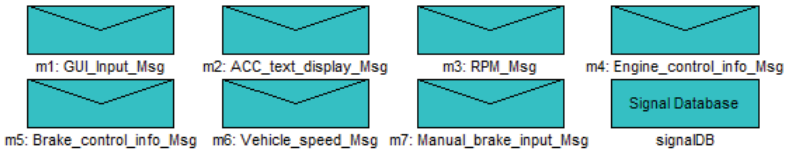


Figure 7.12: Model of CAN messages and signal database in RCM

A high-level architecture of UI, CC, EC and BC nodes in RCM is shown in Figures 7.13, 7.14, 7.15 and 7.16 respectively.

In this Section, we discuss the internal model of only CC node. The details of internal models of the rest of the nodes are presented in Appendix A. The CC node is modeled with four assemblies as shown in Figure 7.14. An assembly in RCM is a container for various software items. The Input_from_Sensors assembly contains an SWC that reads radar sensor values as shown in Figure 7.17. The Input_from_CAN assembly contains three ISWCs, i.e., GUI_Input_Msg_ISWC, Vehicle_speed_Msg_ISWC and Manual_brake_input_Msg_ISWC as depicted in Figure 7.18. These components receive respective messages $m1$, $m6$

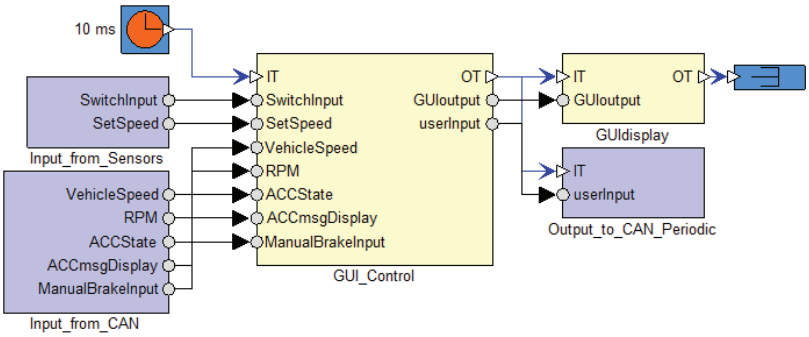


Figure 7.13: Architecture of a User Interface node in RCM

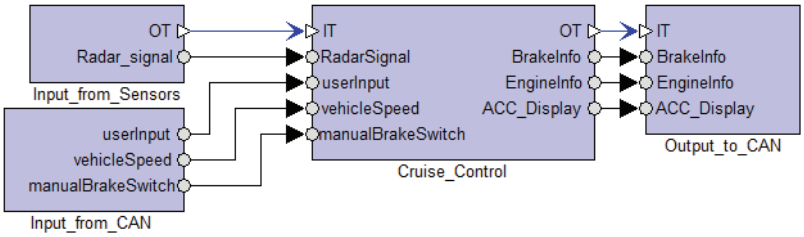


Figure 7.14: Architecture of a Cruise Control node in RCM

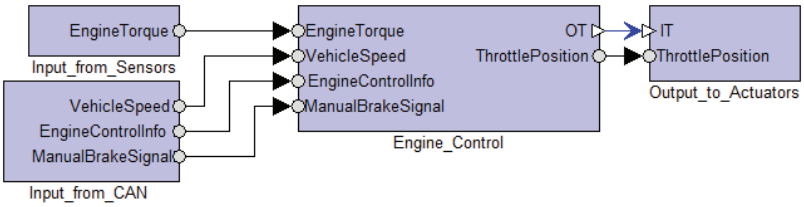


Figure 7.15: Architecture of a Engine Control node in RCM

and $m7$ from CAN network. Similarly, the assembly Output_to_CAN contains three OSWC components as shown in Figure 7.19. These components send messages $m5$, $m4$ and $m2$ to CAN network. The Cruise_Control assembly contains two SWCs: one handles the input and cruise control mode signals while the other processes the received information and produces control mes-

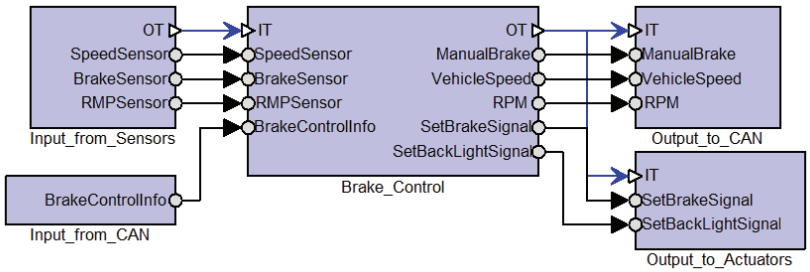


Figure 7.16: Architecture of a Brake Control node in RCM

Table 7.1: Message attributes extracted from the model and corresponding analysis results

| **Msg** | $s_m$ | $P_m$ | $\xi_m$ | $T_m$ $\mu$**Sec** | $C_m$ $\mu$**Sec** | $R_m$ $\mu$**Sec** |
|---|---|---|---|---|---|---|
| $m1$ | 8 | 7 | Periodic | 10000 | 320 | 3860 |
| $m2$ | 8 | 6 | Periodic | 10000 | 320 | 2360 |
| $m3$ | 8 | 4 | Periodic | 10000 | 200 | 2540 |
| $m4$ | 8 | 3 | Sporadic | 10000 | 320 | 7340 |
| $m5$ | 2 | 5 | Sporadic | 10000 | 320 | 6520 |
| $m6$ | 2 | 2 | Periodic | 10000 | 200 | 1200 |
| $m7$ | 1 | 1 | Sporadic | 10000 | 180 | 7500 |

sages for the other nodes. The internal structure of this assembly is shown in Figure 7.20.

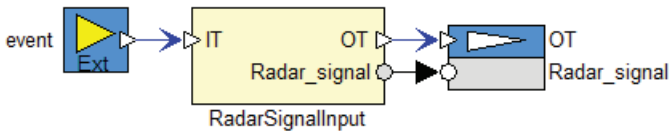


Figure 7.17: Internals of an assembly reading sensors

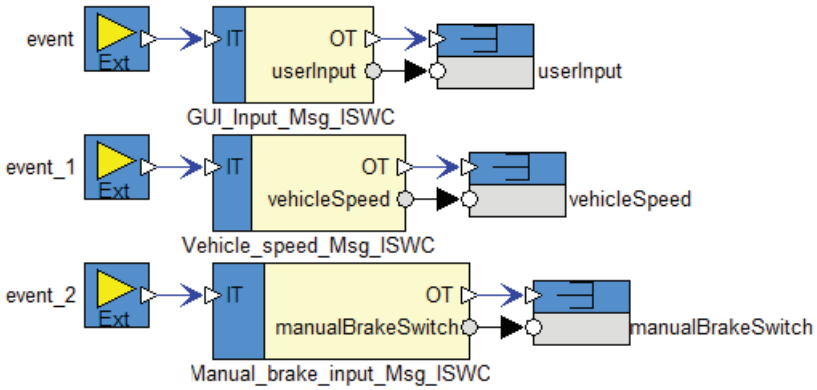


Figure 7.18: Internals of an assembly reading CAN messages

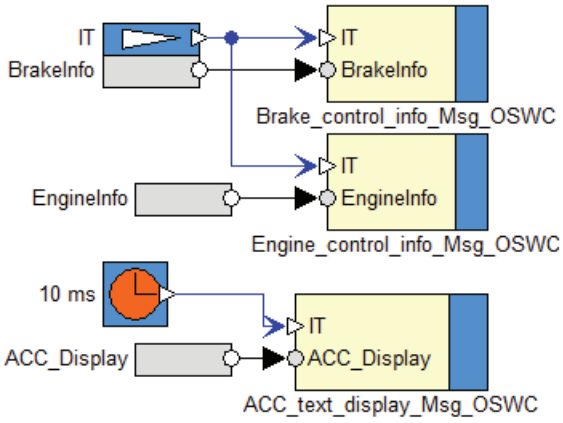


Figure 7.19: Internals of an assembly sending CAN messages

### 7.6.3 Modeling of Deadline Requirements

We specify deadline requirements on four DTs in ACC system using a deadline object in RCM. Upon reading the radar signal, these DTs produce brake and engine actuation signals and display information for the driver. All these transactions have a common initiator, i.e., their first task corresponds to the SWC that reads radar signal and is located in the CC node. The last tasks of first and

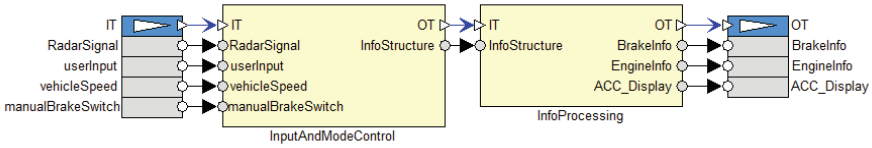


Figure 7.20: SWCs comprising the Cruise Control assembly

second DTs are located in the Brake Control node. These tasks correspond to the SWCs SetBrakeSignal and SetBackLightSignal as shown in Figure. 7.16. The last task of third DT corresponds to SetThrottlePosition SWC and is located in the Engine Control node as shown in Figure. 7.15. The last task of fourth DT corresponds to GUIdisplay SWC and is located in the User Interface node as shown in Figure. 7.13.

### 7.6.4 HRTA of ACC System using HRTA Plug-in

The run-time allocation of all the components in the model of ACC system results in 19 transactions, 36 tasks and 7 messages. Due to lack of space, the extracted timing attributes and detailed analysis results of all transactions and tasks can not be provided. The transmission times ($C_m$) and worst-case response times ($R_m$) of all messages computed by HRTA plug-in are shown in Table 1. The analysis report in Table 2 provides worst-case holistic response times of four DTs (discussed in the previous subsection) using HRTA plug-in. The corresponding deadlines are also shown. The response time of a DT is counted from the activation of the first task to the completion of the last task in the chain. The response time of these four DTs correspond to the production of control signals for brake actuators, back lights, engine throttle actuator and graphical user interface.

## 7.7 Conclusion and Future Work

We presented an implementation of state-of-the-art Holistic Response Time Analysis (HRTA) as a plug-in for the industrial tool suite Rubus-ICE. The implemented analysis is general as it supports the integration of real-time analysis of various networks without a need for changing the holistic analysis algorithm. We discussed and solved several issues that we faced during the implementation, integration and evaluation of HRTA plug-in. The experience gained by

Table 7.2: Analysis Report

| DT | Control Signal Produced by DT | Deadline ($\mu$Sec) | Holistic Response Time($\mu$Sec) |
|---|---|---|---|
| 1 | SetBrakeSignal | 10000 | 6000 |
| 2 | SetBackLightSignal | 10000 | 6500 |
| 3 | SetThrottlePosition | 5000 | 3000 |
| 4 | GUIdisplay | 12000 | 1500 |

dealing with the implementation issues provided a feed back to the component model, for example, feed back on the design decisions for efficient run-time allocation of network interface components. We also discussed the steps that we followed for testing and evaluating HRTA plug-in. We found the integration testing to be a tedious and non-trivial activity. Our experience of implementing, integrating and evaluating HRTA plug-in shows that a considerable amount of work and time is required to transfer complex real-time analysis results to the industrial tools.

We provided a proof of concept by modeling an autonomous cruise control system using component-based development and analyzing it with HRTA plug-in. We believe that most of the implementation issues discussed in this paper are generally applicable when real-time analysis is transferred to any industrial or academic tool suite. Moreover, the contributions in this paper may provide guidance for the implementation of other complex real-time analysis techniques in any industrial tool suite that supports a plug-in framework (for the integration of new tools) and component-based development of DRE systems.

In the future, we plan to implement the analysis of other network communication protocols (e.g., Flexray, switched ethernet, etc.) and integrate them within HRTA plug-in. Another future work could be the implementation of end-to-end latency analysis in Rubus-ICE to support the analysis of multi-rate real-time systems. We also plan to provide a support for asynchronous data-flow using the two different semantics of data-age and reaction described in [33].

## Acknowledgement

This work is supported by Swedish Knowledge Foundation (KKS) within the projects Femmva and EEMDEF, the Swedish Research Council (VR) within project TiPCES, and the Strategic Research Foundation (SSF) with the centre PROGRESS. The authors would like to thank the industrial partners Arcticus Systems and BAE Systems Hägglunds.

## 7.8 Appendix A

### Internal Model of User Interface Node in RCM

The User Interface node is modeled with three assemblies along with two SWCs as shown in Figure 7.13. The GUI_Control SWC handles the input from the sensors and messages from the CAN network. After processing the information, it not only produces information for Graphical User Interface (GUI), but also computes control signals for the other nodes. The GUIdisplay SWC sends the signals (corresponding to updated information) to GUI in the car.

The Input_from_Sensors assembly contains two SWCs as shown in Figure 7.21. One of them reads the sensor values that correspond to the state of the cruise control switch on the steering wheel. The other SWC reads the sensor values that correspond to the vehicle cruising speed set by the driver.

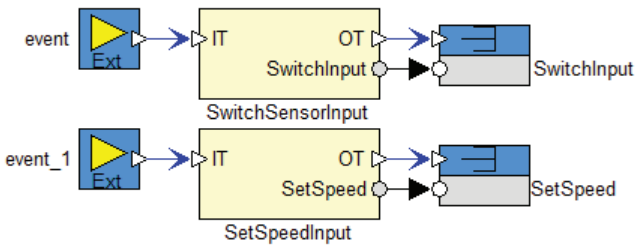


Figure 7.21: User Interface Node: internals of an assembly reading sensors

The Input_from_CAN assembly contains four ISWC components, i.e., Vehicle_Speed_Msg_ISWC, RPM_Msg_ISWC, ACC_text_display_Msg_ISWC and Manual_brake_input_Msg_ISWC as shown in Figure 7.22. These components receive messages $m6$, $m3$, $m2$ and $m7$ from CAN network respectively.

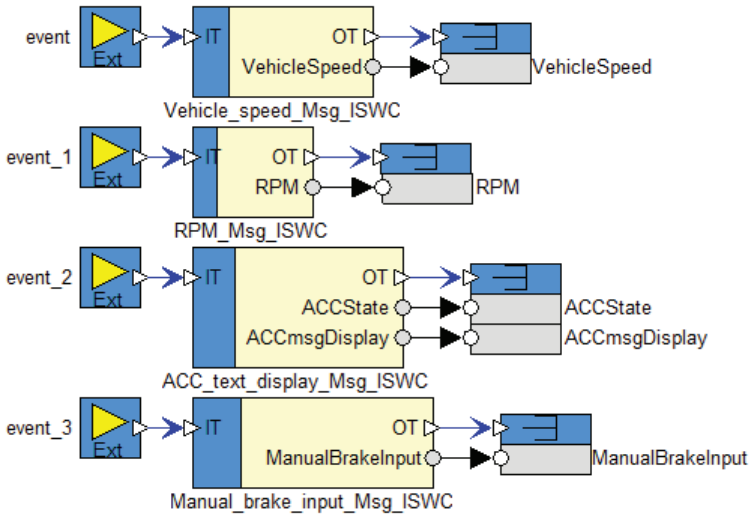


Figure 7.22: User Interface Node: internals of an assembly reading CAN messages

The third assembly, i.e., Output_to_CAN_Periodic sends a message $m1$ to CAN network via the OSWC component as shown in Figure 7.23.

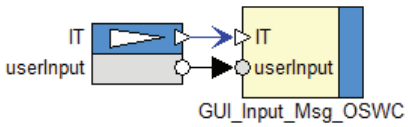


Figure 7.23: User Interface Node: internals of an assembly sending CAN messages

## Internal Model of Engine Control Node in RCM

The Engine Control node is modeled with four assemblies as shown in Figure 7.15. The Input_from_Sensors assembly contains a SWC that reads the sensor values corresponding to the engine torque as shown in Figure 7.24. The Input_from_CAN assembly contains three ISWCs, i.e., Vehicle_Speed_Msg_

ISWC, Engine_control_info_Msg_ISWC and Manual_brake_input_Msg_ISWC as shown in Figure 7.25. These components receive messages $m6$, $m4$ and $m7$ from CAN network respectively. The third assembly, Output_to_Actuators as shown in Figure 7.26, contains the SWC that produces control signals for the engine throttle actuator. The fourth assembly, i.e., Engine_Control as shown in Figure 7.27, contains two SWCs: one handles and processes the inputs from sensors and received messages while the other computes the new position for engine throttle.

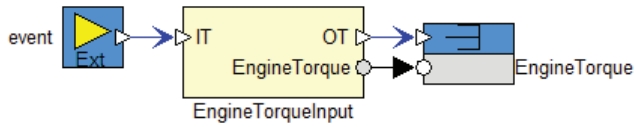


Figure 7.24: Engine Control Node: internals of an assembly reading sensors

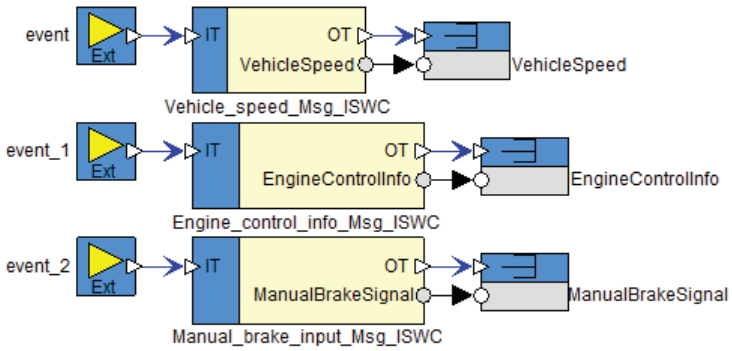


Figure 7.25: Engine Control Node: internals of an assembly reading CAN messages

## Internal Model of Brake Control Node in RCM

The Brake Control node is modeled with five assemblies as shown in Figure 7.16. The Input_from_Sensors assembly contains three SWCs as shown in Figure 7.28. These SWCs read the sensor values that correspond to the values of speed, rpm and manual brake sensors in the vehicle. The Input_from_CAN

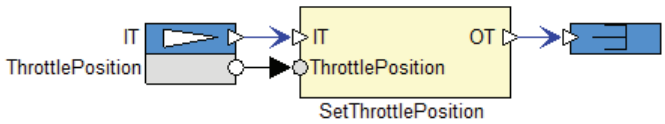


Figure 7.26: Engine Control Node: internals of an assembly producing actuation signals

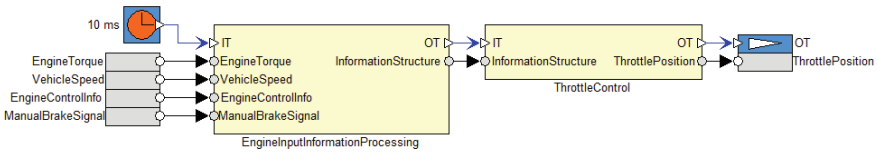


Figure 7.27: Engine Control Node: SWCs comprising the Engine Control assembly

assembly, shown in Figure 7.29, contains the ISWC component Brake_control_info_Msg_ISWC that receives a message $m5$ from CAN network. The third assembly, i.e., Brake_Control as shown in Figure 7.30, contains two SWCs: one handles and processes the inputs from sensors and received messages while the other computes the control signals for brake actuators. The fourth assembly Output_to_CAN contains three OSWC components as shown in Figure 7.31. These components send messages $m7$, $m6$ and $m3$ to CAN network. The fifth assembly, Output_to_Actuators as shown in Figure 7.32, contains the SWCs that produce control signals for the brake actuators and back light controllers.

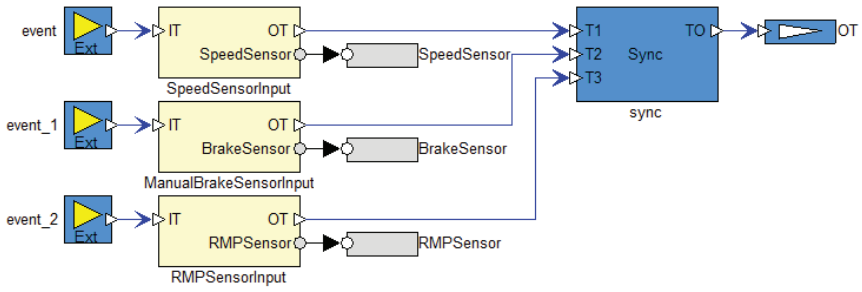


Figure 7.28: Brake Control Node: Internals of an assembly reading sensors

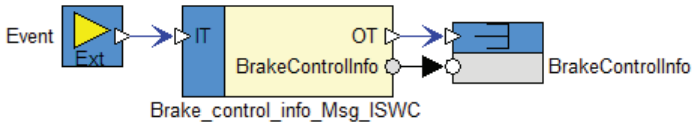


Figure 7.29: Brake Control Node: Internals of an assembly reading CAN messages

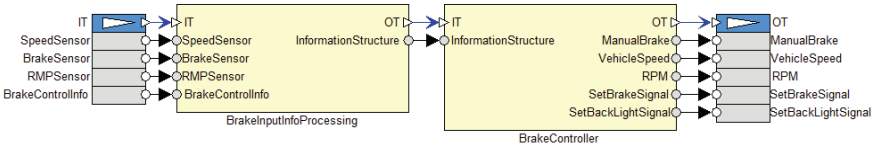


Figure 7.30: Brake Control Node: Internals of Brake Control assembly

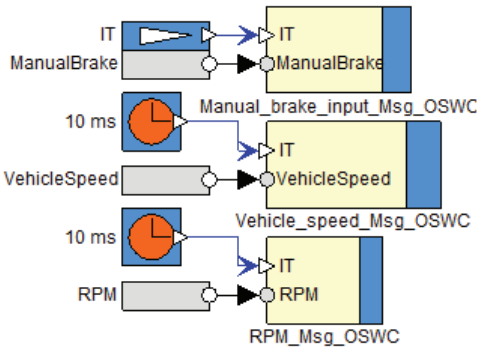


Figure 7.31: Brake Control Node: Internals of an assembly sending CAN messages

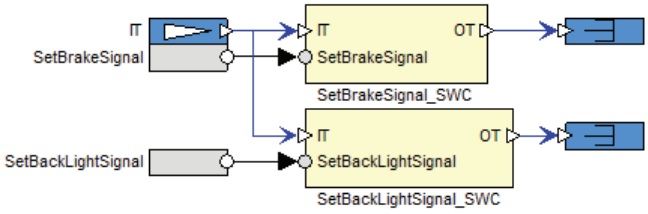


Figure 7.32: Brake Control Node: internals of an assembly producing actuation signals

# Bibliography


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

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

[3] Ken Tindell and John Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocess. Microprogram.*, 40:117–134, April 1994.

[4] K. W. Tindell. Using offset information to analyse static priority preemptively scheduled task sets. Technical Report YCS 182, Dept. of Computer Science, University of York, 1992.

[5] J.C. Palencia and M. Gonzalez Harbour. Schedulability Analysis for Tasks with Static and Dynamic Offsets. *Real-Time Systems Symposium, IEEE International*, page 26, 1998.

[6] Arcticus Systems. http://www.arcticus-systems.com.

[7] K. Hänninen et.al. The Rubus Component Model for Resource Constrained Real-Time Systems. In *3rd IEEE International Symposium on Industrial Embedded Systems*, June 2008.

[8] Saad Mubeen, Jukka Mäki-Turja, Mikael Sjödin, and Jan Carlson. Analyzable Modeling of Legacy Communication in Component-Based Distributed Embedded Systems. In *37th EUROMICRO Conference on Soft-*

*ware Engineering and Advanced Applications (SEAA), 2011*, pages 229 –238, Sep. 2011.

[9] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Extraction of end-to-end timing model from component-based distributed real-time embedded systems. In *Time Analysis and Model-Based Design, from Functional Models to Distributed Deployments (TiMoBD) workshop located at Embedded Systems Week*, pages 1–6. Springer, October 2011.

[10] K. Hänninen et.al. Framework for real-time analysis in Rubus-ICE. In *Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on*, pages 782 –788, 2008.

[11] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *ACM*, 20(1):46–61, 1973.

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

[13] Mikael Nolin, Jukka Mäki-Turja, and Kaj Hänninen. Achieving Industrial Strength Timing Predictions of Embedded System Behavior. In *ESA*, pages 173–178, 2008.

[14] Jukka Mäki-Turja, , and Mikael Nolin. Tighter response-times for tasks with offsets. In *Real-time and Embedded Computing Systems and Applications Conference (RTCSA)*. Springer-Verlag, August 2004.

[15] K.W. Tindell, H. Hansson, and A.J. Wellings. Analysing real-time communications: controller area network (CAN). In *Real-Time Systems Symposium (RTSS) 1994*, pages 259 –263.

[16] Robert Davis, Alan Burns, Reinder Bril, and Johan Lukkien. Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems*, 35:239–272, 2007.

[17] Robert I. Davis, Steffen Kollmann, Victor Pollex, and Frank Slomka. Controller Area Network (CAN) Schedulability Analysis with FIFO queues. In *23rd Euromicro Conference on Real-Time Systems (ECRTS11)*, July 2011.

[18] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Extending schedulability analysis of controller area network (CAN) for mixed (periodic/sporadic) messages. In *Emerging Technologies Factory Automation (ETFA), IEEE 16th Conference on*, sept. 2011.

[19] Mubeen, Saad and Mäki-Turja, Jukka and Sjödin, Mikael. Extending response-time analysis of controller area network (CAN) with FIFO queues for mixed messages. In *Emerging Technologies Factory Automation (ETFA), IEEE 16th Conference on*, pages 1–4, sept. 2011.

[20] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Implementation of Holistic Response-Time Analysis in Rubus-ICE: Preliminary Findings, Issues and Experiences. In *The 32nd IEEE Real-Time Systems Symposium (RTSS), WIP Session*, pages 9–12, December 2011.

[21] MAST–Modeling and Analysis Suite for Real-Time Applications. http://mast.unican.es/.

[22] The Volcano Family. http://www.mentor.com/products/vnd.

[23] Volcano Network Architect (VNA). Mentor Graphics. http://www.mentor.com/products/vnd/communication-management/vna.

[24] Arne Hamann, Rafik Henia, Razvan Racu, Marek Jersak, Kai Richter, and Rolf Ernst. Symta/s - symbolic timing analysis for systems, 2004.

[25] M. Hagner and U. Goltz. Integration of scheduling analysis into uml based development processes through model transformation. In *Computer Science and Information Technology (IMCSIT), Proceedings of the 2010 International Multiconference on*, pages 797 –804, oct. 2010.

[26] RAPID RMA: The Art of Modeling Real-Time Systems. http://www.tripac.com/rapid-rma.

[27] D.G. Schmidt and F. Kuhns. An overview of the Real-Time CORBA specification. *Computer*, 33(6):56 –63, June 2000.

[28] M. Hagner and U. Goltz. Integration of scheduling analysis into uml based development processes through model transformation. In *Computer Science and Information Technology (IMCSIT), Proceedings of the 2010 International Multiconference on*, pages 797 –804, oct. 2010.

[29] Mubeen, Saad and Mäki-Turja, Jukka and Sjödin, Mikael. Tracing event chains for holistic response-time analysis of component-based distributed real-time systems. *SIGBED Review*, 8:48–51, September 2011.

[30] P. Berggren. Autonomous Cruise Control for Chalmers Vehicle Simulator. Master's thesis, Department of Signals and Systems, Chalmers University of Technology, 2008.

[31] Adaptive Cruise Control System Overview. In *Workshop of Software System Safety Working Group*, April 2005. Anaheim, California, USA. Available at: sunnyday.mit.edu/Adaptive_Cruise_Control_Sys_Overview.pdf.

[32] ISO 11898-1. Road Vehicles interchange of digital information controller area network (CAN) for high-speed communication, ISO Standard-11898, Nov. 1993.

[33] N. Feiertag, K. Richter, J. Nordlander, and J. Jonsson. A Compositional Framework for End-to-End Path Delay Calculation of Automotive Systems under Different Path Semantics. In *Compositional Theory and Technology for Real-Time Embedded Systems, 2008. CRTS 2008. Workshop on*, dec. 2008.