Mälardalen University Press Dissertations
No. 160

# COMMUNICATIONS-ORIENTED MODELING AND DEVELOPMENT OF VEHICULAR DISTRIBUTED EMBEDDED SYSTEMS

**Saad Mubeen**

**2014**


School of Innovation, Design and Engineering

Copyright © Saad Mubeen, 2014
ISBN 978-91-7485-151-9
ISSN 1651-4238
Printed by Arkitektkopia, Västerås, Sweden

# Abstract

The model- and component-based development approach has emerged as an attractive option for the development of vehicular distributed real-time embedded systems. Within this context we target challenges related to modeling of legacy network communication, extraction of end-to-end timing models and support for end-to-end timing analysis.

We propose a novel approach for modeling legacy network communication in these systems. By introducing special-purpose components to encapsulate and abstract the communication protocols, we allow the use of legacy nodes and legacy protocols in a component- and model-based software engineering environment. Because an end-to-end timing model should be available to perform the end-to-end response-time and delay analyses, we present a method to extract the timing models from these systems. We also extend the method to various abstraction levels and parts of the development process for the systems. During the models extraction, we identify that the existing worst-case response-time analysis for Controller Area Network (CAN), a widely used real-time network protocol in the vehicular domain, does not support mixed messages. These messages are partly periodic and partly sporadic. They are implemented by some higher-level protocols for CAN used in the industry. We extend the existing analysis which is now applicable to any higher-level protocol for CAN that uses periodic, sporadic and/or mixed transmission.

In order to show the application of our modeling techniques, timing model extraction method and extended analyses; we provide a proof of concept by extending the Rubus Component Model, which is used for the development of software for vehicular embedded real-time systems by several international companies. We also implement the end-to-end response-time and delay analyses along with the extended analysis for CAN in the existing industrial tool suite the Rubus-ICE. Moreover, we implement the extended analysis for CAN in a free tool MPS-CAN analyzer. Further, we conduct automotive-application case studies to validate our methods and techniques.

To my dear uncle and mentor Tariq Zahin

# 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, comments and feedback throughout my studies. I had a great opportunity of learning so many new things from them during our meetings and discussions.

I thank Professor Mikael Sjödin, Dr. Moris Behnam and Dr. Jan Carlson for providing valuable and useful comments on the thesis.

Many thanks to the people from industry who contributed to this thesis. Thank you Dr. Kurt-Lennart Lundbäck, John Lundbäck, Staffan Sandberg, Jimmy Westerlund, Martin Vestin, Henrik Nordin, Peter Wallin and Andreas Ljungberg. Thanks to Hjördis Lundbäck for making many things easier for me during my time as an industrial PhD student at Arcticus Systems.

I attended several courses during my PhD studies. I thank the lecturers and professors including Hans Hansson, Thomas Nolte, Emma Nehrenheim, Mikael Sjödin, Jukka Mäki-Turja, Ivica Crnkovic, Jan Torin, Sasikumar Punnekkat, Kristina Lundqvist and Luis Almeida for their guidance during my studies. I want to also thank other faculty members Paul Pettersson, Lars Asplund, Jan Gustafsson, Björn Lisper, Mats Björkman, Jan Carlson, Damir Isovic, Dag Nyström, Cristina Seceleanu, Moris Behnam, Gordana Dodig-Crnkovic, Mikael Ekström, Andreas Ermedahl, Elisabeth Uhlemann, Radu Dobrin and others. 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 (in a particular order[1] ☺) Svetlana; Elena, Mattias and

[1] Svetlana Girs, Saad Mubeen and Mohammad Ashjaei. A Novel Work Place Proximity-based Ordering Algorithm. In the $187^{th}$ Big Room Coffee-Time (BRCT) Workshop, Västerås, Sweden, August, 2013.

Sara Ab.; Adnan, Danial and Wasif; Eduard, Guillermo and Sara Ab.; Alessio and Kan; Leo, Nesredin and Predrag; Dag; Mohammad and Rafia; Hamid and Matthias; Meng and Sara Af.; Nima; Cristina; Moris; Hang and Raluca; Mobyen and Shahina; Giacomo; Anna and Ning; Andreas G. and Bob; Batu, Fredrik and Nikola; Abhilash, Andreas J., Kaj and Zhou; Hseyin, Patrick and Severine; Hongyu and Gaetana; Barbara and Yue; Radu; Antonio, Federico and Mehrdad; Gabriel, Irfan, Mahnaz and Omar; Ana P., Josip, Juraj and Luka; Adam; Aida, Amine; Andreas H.; Aneta; Etienne; Farhang; Frank; Jagadish; Johan; Jiri; Kivanc; Mikael Å; Olga; Sara D.; Stefan B.; Thomas L.; Tibi and others for all the fun and memories.

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

I would also like the opportunity to thank my friends Imran, Moeen and Shoaib for their support, encouragement and all the fun in the past thirteen years.

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

The work in this thesis has been supported by the Swedish Research Council (VR) within the project SynthSoft; the Swedish Knowledge Foundation (KKS) within the projects FEMMVA and EEMDEF; and the Swedish Foundation for Strategic Research (SSF) with the centre PROGRESS. I would like to thank the industrial partners Arcticus Systems, BAE Systems Hägglunds and Volvo Construction Equipment, Sweden.

Saad Mubeen
Västerås, May, 2014

# List of Publications

## Papers Included in the PhD Thesis[2]

**Paper A** *Communications-Oriented Development of Component- Based Vehicular Distributed Real-Time Embedded Systems*. Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin. In Journal of Systems Architecture (JSA), vol 60, nr 2, Elsevier, 2014.

**Paper B** *Integrating Mixed Transmission and Practical Limitations with the Worst-Case Response-Time Analysis for Controller Area Network*. Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin. Conditionally accepted to the Journal of Systems and Software (JSS), Elsevier, April, 2014.

**Paper C** *Extending Worst-Case Response-Time Analysis for Mixed Messages in Controller Area Network with Priority and FIFO Queues*. Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin. In Journal of Access, IEEE, vol 2, nr 1, 2014.

**Paper D** *Extending offset-based response-time analysis for mixed messages in Controller Area Network*. Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin. In the 18th IEEE conference on Emerging Technologies and Factory Automation (ETFA), September, 2013.
**IEEE Industrial Electronics Society Scholarship Award**.

[2]The included articles have been reformatted to comply with the PhD thesis layout

**Paper E** *MPS-CAN Analyzer: Integrated Implementation of Response-Time Analyses for Controller Area Network*. Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin. In Journal of Systems Architecture (JSA), Elsevier, May, 2014.

**Paper F** *Support for end-to-end response-time and delay analysis in the industrial tool suite: Issues, experiences and a case study*. Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin. In Journal of Computer Science and Information Systems (ComSIS), vol 10, nr 1, 2013.

**Paper G** *Component-Based Vehicular Distributed Embedded Systems: End-to-end Timing Models Extraction at Various Abstraction Levels*. Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin. MRTC report ISSN 1404-3041, Mälardalen Real-Time Research Centre, Mälardalen University, May, 2014 (pending submission as conference contribution).

# Additional Papers, not Included in the PhD Thesis

## Journals

1. *Investigating Techniques to Model Real-Time Network Communication for the Industrial Component Model*. Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin. Information Journal, vol 16, nr 7B, pages 5183–5196, International Information Institute, Japan, July, 2013.

2. *Automated Model Translations for Vehicular Real-Time Systems with Preserved Semantics*. Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin. ACM SIGBED Review, vol 10, nr 2, ACM, July, 2013.

3. *Translating End-to-End Timing Requirements to Timing Analysis Model in Component-Based Distributed Real-Time Systems*. Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin. ACM SIGBED Review, vol 9, nr 4, pages 17-20, ACM, November, 2012.

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

## Conferences and Workshops

5. *Response Time Analysis with Offsets for Mixed Messages in CAN Supporting Transmission Abort Requests*. Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin. Under review.

6. *Implementing and Evaluating Various Response-Time Analyses for Mixed Messages in CAN using MPS-CAN Analyzer*. Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin. Under review.

7. *Towards Communications-Oriented Development of Component-Based Vehicular Distributed Embedded Systems – a Ph.D. Research Proposal*. Saad Mubeen. In 19th International Doctoral Symposium on Components and Architecture (WCOP), Lille, France, June, 2014.

8. *Towards Extraction of Interoperable Timing Models from Component-Based Vehicular Distributed Embedded Systems*. Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin. In 11th IEEE International Conference

on Information Technology : New Generations (ITNG), Las Vegas, USA, April, 2014.

9. *From Modeling to Deployment of Component-Based Vehicular Distributed Real-Time Systems*. Alessio Bucaioni, Saad Mubeen, John Lundbäck, Kurt-Lennart Lundbäck, Jukka Mäki-Turja and Mikael Sjödin. In 11th IEEE International Conference on Information Technology : New Generations (ITNG), Las Vegas, USA, April, 2014.

10. *Demonstrator for Modeling and Development of Component-Based Distributed Real-Time Systems with Rubus-ICE*. Alessio Bucaioni, Saad Mubeen, John Lundbäck, Kurt-Lennart Lundbäck, Jukka Mäki-Turja and Mikael Sjödin. In Open Demo Session of Real-Time Systems located at Real Time Systems Symposium (RTSS), Vancouver, Canada, December, 2013.

11. *Many-in-one Response-Time Analyzer for Controller Area Network*. Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin. In 4th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS), located at the 25th Euromicro Conference on Real-Time Systems (ECRTS13), Paris, France, July, 2013.
**Invited for an extension to the Journal of Systems Architecture**.

12. *End-to-end Timing Challenges in Seamless Tool Chain Development for Vehicular Embedded Real-Time Systems*. Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin. In 25th Euromicro Conference on Real-Time Systems (ECRTS13), WIP, Paris, France, July, 2013.

13. *Supporting Early Modeling and End-to-end Timing Analysis of Vehicular Distributed Real-Time Applications*. Saad Mubeen, Mikael Sjödin, and Jukka Mäki-Turja. In the Real-Time and Distributed Computing in Emerging Applications (REACTION) Workshop located at 33rd IEEE Real-Time Systems Symposium (RTSS), Puerto Rico, USA, December, 2012.
**Invited for an extension to the Journal of Systems Architecture**.

14. *Worst-Case Response-Time Analysis for Mixed Messages with Offsets in Controller Area Network*. Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin. In 17th IEEE Conference on Emerging Technologies and Factory Automation (ETFA), Krakow, Poland, September, 2012.

15. *Extending Response-Time Analysis of Mixed Messages in CAN with Controllers Implementing Non-Abortable Transmit Buffers*. Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin. In 17th IEEE Conference on Emerging Technologies and Factory Automation (ETFA), WIP, Krakow, Poland, September, 2012.

16. *Response Time Analysis for Mixed Messages in CAN Supporting Transmission Abort Requests*. Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin. In 7th IEEE International Symposium on Industrial Embedded Systems (SIES), WIP, pages 291-294, Karlsruhe, Germany, June, 2012.

17. *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 9th IEEE International Workshop on Factory Communication Systems (WFCS), pages 23-32, Lemgo, Germany, May, 2012.

18. *Implementation of End-to-End Latency Analysis for Component-Based Multi-Rate Real-Time Systems in Rubus-ICE*. Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin. In 9th IEEE International Workshop on Factory Communication Systems (WFCS), WIP, pages 165-168, Lemgo, Germany, May, 2012.

19. *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. In 19th IEEE conference on Engineering of Computer Based Systems (ECBS), pages 210-221, Novi Sad, Serbia, April, 2012.
    **Among Best Selected Papers, invited for an extension to the Journal of Computer Science and Information Systems**.

20. *Towards Modeling and Holistic Timing Analysis of Industrial Component-Based DRE Systems*. Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin. In 19th IEEE conference on Engineering of Computer Based Systems (ECBS), pages 283-292, Novi Sad, Serbia, April, 2012.

21. *Implementation of Holistic Response-Time Analysis in Rubus-ICE: Preliminary Findings, Issues and Experiences*. Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin. In 32nd IEEE Real-Time Systems Symposium (RTSS), WIP, pages 1-4, Vienna, Austria, December, 2011.

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

23. *Extending Schedulability Analysis of Controller Area Network for Mixed (Periodic/Sporadic) Messages*. Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin. In 16th IEEE Conference on Emerging Technologies and Factory Automation (ETFA), Toulouse, France, September, 2011.

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

25. *Analyzable Modeling of Legacy Communication in Component-Based Distributed Embedded Systems*. Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin. In 37th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pages 229-238, Oulu, Finland September, 2009.

26. *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 6th International Conference on Embedded and Multimedia Computing (EMC), Lecture Notes in Electrical Engineering (LNEE), Springer, pages 441-458, August, 2011.
**Invited for an extension to the INFORMATION Journal**.

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

28. *Designing Efficient Source Routing for Mesh Topology Network on Chip Platforms*. Saad Mubeen and Shashi Kumar. In 13th Euromicro Conference on Digital System Design, Architectures, Methods and Tools (DSD), pages 181-188, September, 2010.

29. *High Precision Response Time Analysis of Tasks with Precedence Chains*. Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin. In 22nd Euromicro Conference on Real-Time Systems (ECRTS), WIP, Brussels, Belgium, July, 2010.

## Book chapters

30. *Extracting End-to-End Timing Models from Component-Based Distributed Embedded Systems*. Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin. Embedded Systems Development, Embedded Systems series, ISBN: 978-1-4614-3878-6, Springer New York, pages 155-169, 2014.

31. *How to Design Source Routing for Mesh Topology Network on Chip?*. Saad Mubeen and Shashi Kumar. Networks - Emerging Topics in Computer Science, ISBN: 978-1-461098-713, iConcept Press, March, 2013.

## Technical reports

32. *Extending Response-Time Analysis for Mixed Messages with Offsets in Controller Area Network*. Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin. Technical Report, Mälardalen University, Sweden, April, 2013, http://www.es.mdh.se/publications/2924-.

33. *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.

34. *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 vehicular distributed real-time embedded systems[1]. By introducing special-purpose software components to encapsulate and abstract the communication protocols in the systems, we allow the use of legacy nodes and legacy protocols in a component- and model-based software engineering environment. The proposed approach is aimed to support the state-of-the-practice development of the systems. End-to-end timing analysis can be used to validate timing requirements (without exhaustive testing ) specified on the systems. In order to perform the end-to-end timing analysis, an end-to-end (or holistic) timing model should be extracted from the system. We present a method to automatically extract such models from the composition of components. We also take a step towards broadening the scope and usability of our approach by extracting the timing models from the systems at a higher abstraction level. Moreover, we validate our methods and techniques by conducting industrial case studies.

## 1.1 Background

### 1.1.1 Embedded System and Embedded Software

An embedded system is a microprocessor-based system that is designed to perform a dedicated functionality by means of hardware and software [1]. Often,

[1] Throughout this thesis, we use the terms *system* or *application* to refer to a software component based, distributed, real-time, embedded system or application

embedded systems interact with their environment through sensors and actuators. They mostly remain hidden in their applications because they are embedded inside larger systems 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, smart phones and modern premium cars. Their applications span over many domains such as automotive, aerospace, consumer electronics, biomedical, 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 the 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.

### 1.1.2 Real-time Embedded System

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 a soft real-time system, infrequent deadline misses can be tolerated. For example, electronic window control system in a car can be a soft real-time system. On the other hand, missing a deadline in a hard real-time system can result in system failure. In a hard real-time system, 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.

### 1.1.3 Model- and Component-based Development

In order to, for example, capture system requirements early during the development; handle the complexity of embedded software; lower the development cost; enable faster turn-around times in early design phases; allow reusability; provide possibilities to automatically perform timing analysis, derive test cases and generate code; and support modeling at higher levels 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 and other design artifacts. Whereas, CBSE facilitates the development of large software systems by integration of software components. It raises the level of abstraction for software development and aims to reuse software components and their architectures. An abstraction level provides a complete definition of the system for a given purpose during the development process. There is a great interest for bringing these development techniques in the embedded systems industry [5, 6].

### 1.1.4 Distributed Real-time Embedded System

Most of the vehicular functions are developed as distributed real-time embedded systems. In these systems, the functionality is distributed over many nodes. Whereas, the nodes can be connected to five or more types of networks [7]. The software development of these systems is much more complex compared to uniprocessor embedded real-time systems due to 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 distributed real-time embedded 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, e.g., adaptive cruise control, intelligent parking assistance, brake-by-wire, steer-by-wire and anti-lock braking. These functions may be allocated to about 70 to 100 Electronic Control Units (ECUs) that may be connected by more than five different types of buses (or networks) [8]. Similarly, the software in a truck can consist of as many as 2000 software functions that may be distributed over 45 ECUs [9].

When MBE and CBSE are used for the development of these systems, modeling of communication infrastructure arises as a challenge. In the industry, the

systems are built often using legacy (sub) systems. Legacy systems are previously developed systems and use predefined rules for communication. Furthermore, the systems are often expected to use legacy network protocols for real-time communication. A component technology for the development of the systems should abstract application software from the communication infrastructure. Nevertheless, it should support the modeling and analysis of legacy communications and legacy systems. It should also support accurate timing analysis of distributed functions.

### 1.1.5 Response-time and Delay Analysis

The safety-critical nature of many distributed real-time embedded 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 [10, 11, 12] have been developed by the research community. Response-Time Analysis (RTA) [10, 11, 12, 13] 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 network respectively. The holistic or end-to-end response-time and delay analyses [14, 15] are schedulability analysis techniques which calculate upper bounds on the response times and delays of event chains that are distributed over more than one node in the system. The end-to-end timing model of the system should be available to perform the analysis. Ideally, a component technology for the development of the systems should support automatic extraction of such timing model.

### 1.1.6 Controller Area Network (CAN)

There are a number of real-time network protocols used in these systems. Among them, Controller Area Network (CAN) [16] is one of the widely used, especially in the automotive domain. In 2003, the International Organization for Standardization (ISO) standardized CAN in ISO 11898-1 [17]. According to CAN in Automation (CiA) [18], more than two billion CAN controllers have been sold until today. Out of this huge number, approximately 80% CAN controllers have been used in the automotive domain. For example, there can be as many as 20 CAN networks used in heavy vehicle architectures such as a

modern truck, while the number of CAN messages transmitted over these networks can be over 6000 [9]. These facts and figures indicate the popularity of CAN in the automotive domain. It is also used in other domains such as industrial control, medical equipments, maritime electronics, production machinery, and many others [19]. It is a multi-master, event-triggered, serial communication bus protocol supporting bus speeds of up to 1 Mbit/s. CAN with Flexible Data-rate (CAN FD) [20] is a new protocol based on CAN that can achieve bus speed of more than 1 Mbit/s. In this thesis, we focus only on CAN and some of its higher-level protocols which are developed for various industrial applications. These include CAN Application Layer (CAL) [21], CANopen [22], Hägglunds Controller Area Network (HCAN) [23], AUTOSAR communications [24], CAN for Military Land Systems domain (MilCAN) [25].

## 1.2 Problem Statement and Research Challenges

The model- and component-based development of software architecture for real-time embedded systems in modern vehicles has had a surge in the last few years. The majority of existing model- and component-based development approaches allow for *structural* and *functional* modeling. They do not support *execution modeling*. The structural modeling is concerned with the structure definition of requirements and higher-level architectural objects. The functional modeling refers to the structured way of representing software functions for the system to be modeled. Whereas, the execution modeling is concerned with the modeling of run-time properties and requirements (e.g., end-to-end deadlines, jitter, etc.) of software functions. The modeling of systems should extend down to the execution level to allow precise control of resource utilization and that timing requirements are not violated. However, providing such a modeling support for distributed real-time embedded systems is very challenging because the functionality in these 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 challenges during the development of the systems in the industry is to model and express timing-related information and perform timing analysis [26, 27, 28].

One way to deal with these challenges is to use a component technology that allows model- and component-based development of the 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 support the expression of timing related information and facilitate the identification of timing errors during the development by rendering the modeled application for end-to-end timing analysis with ease and unambiguity.

However, building such a component technology raises many challenges. One of the main reasons behind these challenges is that the development process for these 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 [4, 5, 29, 30, 31, 32]. 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 the systems is often not practiced in the industry, especially in the vehicular domain. In the industry, most often, bottom-up development approach is used as a lot of information, artifacts and solutions are reused from other projects. The infrastructure and platform (e.g., machine, types of ECUs and buses) for the system to be developed is already known. The traditional process for the development of these systems in the industry starts with designing the bus (or network) communication. In the early stage of industrial development process, usually the focus is on finding the answers to the following questions. How many busses will there be in the system? Which nodes will be connected to which bus? How many messages will there be in the system? Which messages will be sent by each node? After finding answers to these questions, the focus is shifted towards the development of functions. Thus, communications-oriented development process is used for the systems and constitutes the state of the practice. By communications, we mean the technology or infrastructure employed in doing communication. This type of development process for the systems is the main focus of this thesis.

Within this context, we target the challenges concerned with the modeling of real-time network communication and support for end-to-end timing analysis. One such challenge is to support the modeling of legacy network communication and allow the use of legacy nodes in the systems. In order to ensure that the 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 the systems should support state-of-the-art real-time analyses such as end-to-end response-time and delay analyses. In order to perform the analysis, the end-to-end timing model of the system should be available. The extraction of the end-to-end timing model from the systems is also a challenge that

we target. The end-to-end timing analysis should be able to analyze end-to-end times in the systems that use realistic CAN controllers and communication protocols. The existing response-time analysis techniques do not support mixed transmission patterns that are implemented by some higher-level protocols for CAN used in the automotive industry. Extension of the existing analyses to support these protocols is another challenge that we deal with in this thesis.

While the introduction of models into the development of the systems increases efficiency in some parts of the engineering process, the models are also cause of novel concerns. In particular, mismatch between structural and semantic assumptions in modeling languages used at various abstraction levels and in different phases during the development of the systems cause large problems when design artifacts are transformed between modeling languages.

In the industry, productivity is hampered by incompatible tools and file-formats, in conjunction with the need for non-trivial, manual and tedious translations between different model-formats. Moreover, these translations are done in ad hoc fashion making the result of the translation unpredictable and potentially altered in terms of semantics. There is a strong need to investigate how to effectively and efficiently work with existing modeling languages for vehicular embedded real-time systems. In this context, we focus on the challenges that are faced when the end-to-end timing models are extracted from the systems that are developed and annotated with timing information with more than one methodology and language at various abstraction levels and development phases. Particularly, we target TIMMO methodology [26] and TADL2 language [33] at higher abstraction level and the Rubus Component Model [34] at lower abstraction level.

The research problem addressed in this thesis can be refined and formulated into the following research challenges.

1. How to support the modeling of legacy network communication and using legacy nodes during the model- and component-based development of distributed real-time embedded systems?

2. How to extract end-to-end timing models from component-based distributed real-time embedded systems that are built using the communications-oriented development processes?

2.5 While dealing with the first two research challenges, we discovered that the existing worst-case response-time analysis for CAN does not support mixed messages. These messages are partly periodic and partly sporadic. They are implemented by some higher-level protocols for CAN used in

the industry. This sub challenge can be formulated as follows. How to analyze end-to-end times in the systems that use realistic CAN controllers and protocols?

3. Given the timing model extraction method developed in response to the second challenge, how can it be used at a higher abstraction level to extract the end-to-end timing models from the systems?

# 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 vehicular distributed real-time embedded systems. The contributions in this thesis are organized in five parts. In the first part, we introduce a new technique for modeling legacy network communication. In the second part, we present a method to extract end-to-end timing models from the systems. In the third part, we identify a need for the extension of existing response-time analysis for CAN, and accordingly, we present the extended analysis. In the fourth part, we provide a proof-of-concept implementation of the techniques developed in previous three parts. We also discuss the challenges faced and corresponding solutions proposed during the implementation of these techniques in an industrial setup. Moreover, we perform case studies to validate our techniques. Finally, in the fifth part, we broaden the scope and usability of our techniques by providing a method to extract the end-to-end timing models from the systems at a higher abstraction level. We provide a summary of these contributions in the following sections.

**Personal Contribution.** The research work presented in these contributions is done in collaboration with my supervisors Prof. Mikael Sjödin and Dr. Jukka Mäki-Turja. I am the main contributor and first author of all the papers.

## 2.1 Contribution 1: Modeling of Legacy Network Communication

This contribution addresses first research challenge. We introduce a new approach for modeling real-time network and legacy communication in component-based distributed real-time embedded systems. In order to show usability of our modeling approach, we implement it by extending the existing industrial component model, i.e., the Rubus Component Model (RCM) [34]. By introducing special-purpose components to encapsulate and abstract the communication protocols in the 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 is able to not only model real-time network communication, but also support state-of-the-practice development of the 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. A conceptual example of a two-node distributed real-time application modeled with new components and objects namely *Out software Circuit (OSWC)*, *In Software Circuit (ISWC)* and *Network Specification (NS)* is depicted in Figure 2.1. Although RCM is selected for the proof-of-concept implementation, the proposed extensions should be generally applicable for the extension of several component technologies for the development of the systems that use a pipe-and-filter style for components interconnection such as ProCom [29] and COMDES-II [30]. This contribution is discussed in detail in Paper A.

## 2.2 Contribution 2: Extraction of End-to-end Timing Models

This contribution addresses second research challenge. The end-to-end response-time and delay analyses are important activities during the development of the systems. In order to perform the analysis, the end-to-end timing models should be extracted from the systems. The extraction of these models can be

challenging because the design and analysis models are usually built using different meta-models. We present a method to automatically extract such models from the component-based systems to facilitate end-to-end response-time and delay analyses. This method is built upon the modeling approach that we discussed in the first contribution. We discuss and solve the issues concerning the model extraction such as extraction of unambiguous timing and linking information from all nodes and networks in the system; and linking of event chains in distributed transactions. For example, the linking information is captured in the NS object shown by the curved arrows in Figure 2.1. The model extraction method and the solutions of encountered problems may be applied to several component technologies that use a pipe-and-filter style for components interconnection. The end-to-end timing model that we considered is also general as it incorporates several real-time network protocols used in the automotive domain. To show the applicability of our approach, we demonstrate the implementation of the model extraction method in the analysis framework of an existing industrial tool suite the Rubus-ICE [35]. This contribution is discussed in detail in Paper A.

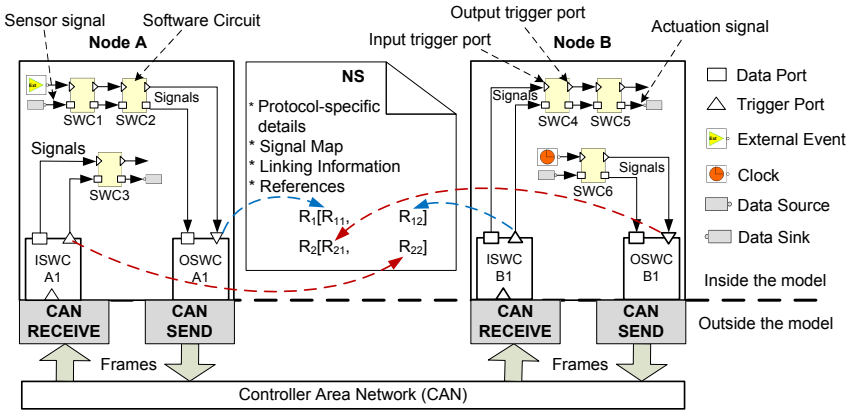


Figure 2.1: Example of a two-node distributed real-time application modeled with the new approach

## 2.3 Contribution 3: Extension of the Worst-case Response-time Analysis for Controller Area Network

This contribution addresses a part of second research challenge identified as sub challenge 2.5. To analyze communications in the 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 higher-level protocols. While attacking the first two research challenges, we identified that the existing Worst-Case Response-Time (WCRT) analysis for CAN does not support common message transmission patterns which are implemented by some higher-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 higher-level protocols based on CAN such as CANopen, HCAN and AUTOSAR communications that support the transmission of mixed messages as well. A mixed message is partly periodic and partly sporadic, i.e., it 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 the systems employing higher-level protocols based on CAN, we identify the need for the extension of the existing WCRT analysis.

We extend the existing WCRT analysis for CAN to support mixed messages. We also extend the analysis for mixed messages while taking into account limitations in CAN controllers such as abortable and non-abortable transmit buffers. These extended analyses are discussed in detail in Paper B. We further extend the WCRT analysis for mixed messages in CAN where some CAN controllers use priority queues while others use First In First Out (FIFO) queues. The extended analysis is discussed in detail in Paper C. Moreover, we extend the WCRT analysis for mixed messages in CAN that are scheduled with offsets and have arbitrary jitter and deadlines. The extended analysis is discussed in detail in Paper D. All these extended analyses are applicable to any higher-level protocol for CAN that uses periodic, sporadic and mixed transmission of messages. Figure 2.2 graphically depicts the relationship between the existing and extended WCRT analyses for CAN.

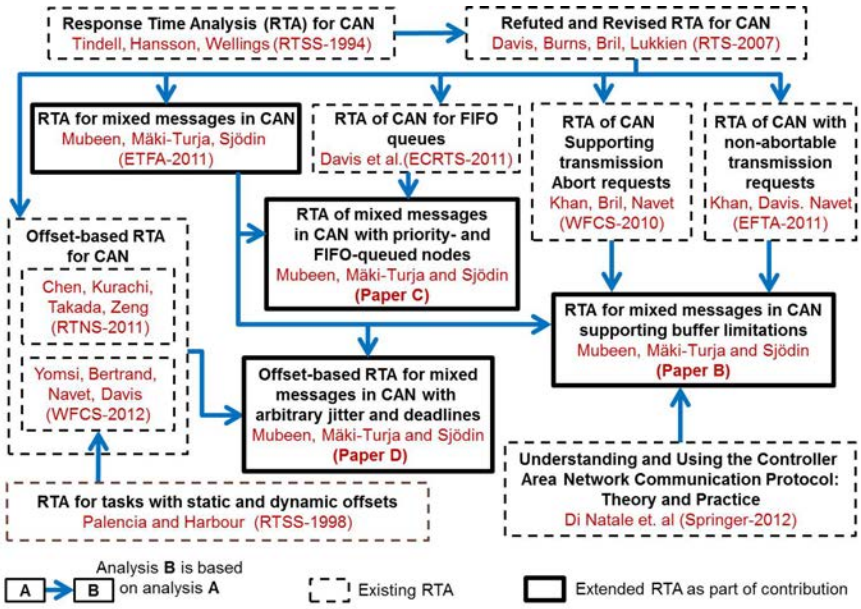


Figure 2.2: Graphical representation of the Response Time Analysis (RTA) and its extensions as part of the contribution

## 2.4 Contribution 4: Proof-of-concept Implementation

In this contribution we validate our solutions to the first two research challenges. We develop a free graphical tool namely MPS-CAN[1] analyzer and implement the extended WCRT analyses for CAN in it. With the implementation of the analyses, the tool is able to perform WCRT analysis for CAN while taking into account mixed messages, messages scheduled with offsets, messages with arbitrary jitter and deadlines, various queueing policies (e.g., priority- or FIFO-based), and limitations of transmit buffers in CAN controllers (e.g., abortable or non-abortable). The tool can also analyze network communications in heterogeneous CAN-based systems that may consist of different types of ECUs supplied by different suppliers. We conduct a case study of a hetero-

[1] https://github.com/saadmubeen/MPS-CAN.

geneous application from the automotive domain to show the usability of the tool. Moreover, we perform a detailed evaluation of the implemented analyses. This part of the fourth contribution explicitly addresses sub challege 2.5 and is discussed in detail in Paper E.

In order to transfer the new modeling technique, timing model extraction method 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 the 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 discuss the implementation of the end-to-end response-time and delay analyses as two individual plug-ins for the existing industrial tool suite Rubus-ICE. The tool suite is used for the development of software for vehicular embedded systems by several international companies. As part of the end-to-end timing analysis, we implement the existing as well as some of the extended analyses for CAN. The implemented analysis is general as it supports the integration of response-time analysis of various networks without a need for changing the algorithm for end-to-end timing analysis. We describe and solve the problems encountered and highlight the experiences gained during the process of implementation, integration and evaluation of the analysis plug-ins for Rubus-ICE. 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 the systems. Finally, we provide a proof of concept for all modeling approaches and extended analysis discussed in the first three contributions by modeling an automotive application using the extended Rubus Component Model, and analyzing it with the implemented analysis in Rubus-ICE. This contribution is discussed in detail in Paper F.

## 2.5 Contribution 5: Extraction of End-to-end Timing Models at a Higher Abstraction Level

We take a step towards broadening the scope and usability of our techniques in the first three contributions. The majority of existing model- and component-based development approaches for the systems support the extraction of end-to-end timing models at an abstraction level and development phase that is

close to the system implementation [36, 29, 37, 38, 30, 32]. Furthermore, the high-level timing analysis provided by existing approaches does not provide high precision and is often not based on actual implementation of the system. As a result, a high-precision end-to-end timing analysis cannot be performed at higher abstraction levels. In the past few years, one of the focuses of several large EU research projects, that involve both academia and industry, has been on supporting the timing analysis at various abstraction levels and development phases [26, 39, 28].

Extraction of the timing model at higher abstraction levels is challenging mainly because not all timing information that is required to be captured in the timing model is available. Moreover, mismatch and incompatibilities among various methodologies, languages and tools that are used in different development phases also add to the complexity of extracting the timing model. Since complete timing information may not be available at higher levels, the timing analysis results may not represent accurate timing behavior of the final system. However, these results can provide useful information that can guide further model refinement and implementation.

We envision the extraction of end-to-end timing model and performing high-precision end-to-end timing analysis at higher levels of abstraction to be state of the practice in the future. We believe, timing information will be formally modeled at higher abstraction levels in the vehicular industry. In that case, we need to extract the specified timing information at higher abstraction levels and connect it to the implementation to generate the end-to-end timing model. Otherwise, it can be too late to extract the timing model at lower abstraction levels that are close to system implementation.

We have experienced that timing information is modeled at higher abstraction levels in the vehicular industry. This may be carried out using SysML language [40]. However, it is done mostly in an informal and textual way; which cannot be used for any formal timing analysis. Today, TADL2 provides the only viable formal method for modeling of timing information using timing constraints at various abstraction levels. In order to extract a complete end-to-end timing model and perform a high-precision timing analysis, TADL2 has to be combined with a lower abstraction level execution modeling approach such as RCM. We hope the industry will start using TADL2. If they do so, we can reuse that information to perform high-precision end-to-end response-time and delay analyses at a higher abstraction level.

We extend our model extraction method (from the second contribution) to support the extraction of end-to-end timing models from the systems at a higher abstraction level. At the higher level, the method extracts timing infor-

mation from system models that are developed with EAST-ADL [41] using the TIMMO methodology; and annotated with timing information using TADL2. At the lower level, the method exploits RCM and its tool suite Rubus-ICE to extract the timing information that cannot be clearly specified at the higher level, e.g., trigger paths in distributed chains. However, it is not straightforward to combine TADL2 with RCM due to several challenges such as unambiguous transformation of TADL2 timing constraints in RCM; and unambiguous extraction of control and data paths at the higher level. The main focus of this contribution is to attack these challenges. Hence, we provide an interpretation of TADL2 timing constraints in RCM. Moreover, we propose extensions in RCM for unambiguous transformation of TADL2 timing constraints. This contribution is discussed in detail in Paper G.

A relationship among the research challenges, contributions and publications included in this thesis is depicted in Figure 2.3.

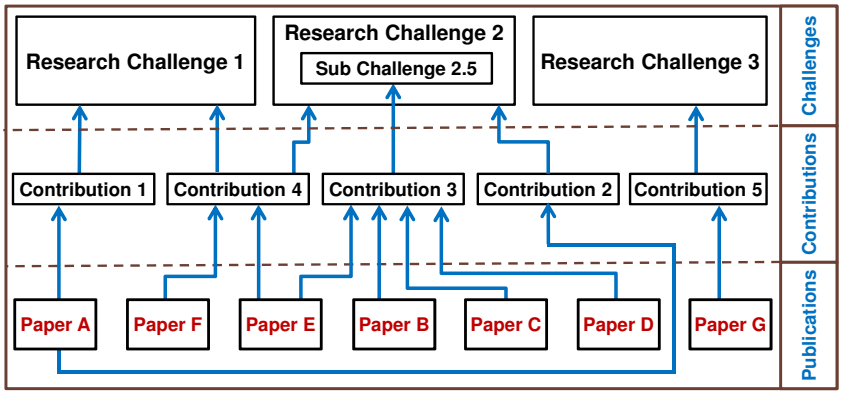


Figure 2.3: Graphical relationship among the research challenges, contributions and publications

## 2.6 Discussion

We select Rubus (RCM and Rubus-ICE) to provide the proof-of-concept implementation for our new modeling techniques and extended analysis for several reasons such as its existing support for structural, functional and execution modeling; capability for developing predictable and analyzable control func-

tions with a support for modeling real-time properties and requirements; separation of interconnections between the functions in terms of data flow and control flow; and automatic generation of run-time framework.

With the proposed extensions, RCM along with Rubus-ICE can be considered a suitable choice for the industrial development of the systems for many reasons. For example, it complements the structural and functional modeling with the execution modeling; supports communications-oriented development process; enables the modeling of legacy communication and legacy systems; models and specifies the timing related information easily; has a small run-time footprint (timing and memory overhead); implements the state-of-the-art research results; and has a strong support for development and analysis tools.

## 2.7 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 that use a pipe-and-filter style for components interconnection. The end-to-end timing model that we considered is also general as it incorporates several real-time network protocols used in the automotive domain. The extended analysis supports common message transmission patterns that are implemented by several higher-level protocols used in the industry today. Further, the extended analysis considers various queueing policies and limitation in practical CAN controllers. The analysis engines support 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 and component-based development of the systems.

The new release of RCM and Rubus-ICE (Version 4.0), which is already in the industrial use, incorporates the contributions and results presented in this thesis. Moreover, MPS-CAN analyzer (Version 0.3) includes the extended analyses for CAN.

# 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 vehicular distributed real-time embedded control systems.

In response to the first research challenge, we proposed a new approach for modeling legacy network communication in the systems. The proposed approach abstracts the implementation and configuration of communications in the systems. It explicitly enables the communication capabilities of a node, but 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. It also enables adaptation of a node when communication rules change without affecting its internal component design. As a solution to the second research challenge, we presented a method to extract end-to-end timing models from the systems that are developed using the above modeling approach. We also discussed and resolved various issues that are faced during the model extraction. The purpose of extracting the end-to-end timing models is to support end-to-end response-time and delay analyses of the systems. In response to the third research challenge, we broaden the scope of our techniques and methods by developing a method to extract the end-to-end timing models at a higher abstraction level and early parts of the development process.

We believe, these techniques may be suitable for several other model- and component-based development technologies for distributed embedded systems

that use a pipe-and-filter style for components interconnection. Moreover, they 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 solutions to the first two research challenges, we identified a need for the extension of existing worst-case response-time analysis for CAN to support mixed messages that are partly periodic and partly sporadic. These messages are implemented by several higher-level protocols for CAN that are used in the industry today. We extended the existing analysis which now supports mixed messages while taking into account offsets, arbitrary jitter and deadlines, various queueing policies (e.g., priority or FIFO), and limitations of transmit buffers in the CAN controllers (e.g., abortable or non-abortable). The extended analyses are applicable to any higher-level protocol or commercial extension of CAN that uses periodic, sporadic and mixed messages. The extended analyses are able to analyze network communications in heterogeneous CAN-based systems which may consist of different types of ECUs (with respect to queueing policies and buffer limitations) supplied by different Tier-1 suppliers. We implemented the extended analyses for CAN in a free tool MPS-CAN analyzer. Using the tool we performed comparative evaluation of the extended analyses in detail.

We provided a proof-of-concept implementation of our modeling and analysis approaches by extending the existing industrial component model RCM, implementing the extended end-to-end response-time and delay analyses in an industrial tool suite the Rubus-ICE, and conducting automotive-application case studies. The analysis engines that we provide are able to predict important execution characteristics of the system such as holistic response times and delays without a need for tedious and expansive testing.

We believe, the tools implementing our modeling techniques and extended analyses may prove helpful for the software development organizations in the vehicular domain to decrease the costs for software development, configuration and testing. Although we selected the platform from vehicular domain for the proof-of-concept implementation, our techniques and methods can be equally used in any other domain where distributed control functionality is involved.

## 3.2 Future Work

In the future we plan to conduct industrial case studies to show the usability of the timing model extraction method at various abstraction levels. An interesting future research direction is to investigate and develop patterns that allow transformation between several domain-specific modeling technologies in the vehicular domain. In this context, we plan to bridge the semantic gap between standard models and languages such as AUTOSAR and EAST-ADL; and/or proprietary languages such as Simulink, Statemate and RCM. It would also be interesting and useful to facilitate the exchange of timing analysis models and tools between RCM and several other component models and tools, including AUTOSAR-based tool chain. Another future work is to develop an optimized offset assignment method for the systems that contain periodic as well as mixed messages. In the future, the end-to-end response-time and delay analyses plug-ins can be expanded by implementing and integrating the analysis of other network communication protocols (e.g., Flexray and switched ethernet) within the end-to-end analysis algorithms discussed in this thesis.

# 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.

[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] In-Vehicle Networking Solutions, Renesas, available at http://www.-renesas.eu, accessed May, 2014.

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

[9] Peter Thorngren. Keynote Talk: Experiences from EAST-ADL Use, EAST-ADL Open Workshop, Gothenberg, Oct., 2013.

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

[11] 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.

[12] 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.

[13] M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, 1986.

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

[15] 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 *Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS)*, Dec. 2008.

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

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

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

[19] Marco Di Natale, Haibo Zeng, Paolo Giusto, Arkadeb Ghosal. *Understanding and Using the Controller Area Network Communication Protocol*. Springer, 2012.

[20] Robert Bosch GmbH. CAN with Flexible Data-Rate (CAN FD), White Paper, Ver. 1.1. 2011.

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

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

[23] Hägglunds Controller Area Network (HCAN), Network Implementation Specification. *BAE Systems Hägglunds, Sweden (internal document)*, Apr. 2009.

[24] Requirements on Communication, Rel. 4.1, Rev. 3, Ver. 3.3.1, March, 2014. www.autosar.org/download/R4.1/AUTOSAR_SRS_COM.pdf, accessed on May 05, 2014.

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

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

[27] Mastering Timing Information for Advanced Automotive Systems Engineering. In the TIMMO-2-USE Brochure, 2012. Available at: http://www.timmo-2-use.org/pdf/T2UBrochure.pdf.

[28] CRYSTAL - CRitical sYSTem engineering AcceLeration, http://www.-crystal-artemis.eu, accessed May, 2014.

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

[30] 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, Aug. 2007.

[31] CHESS Project, CHESS consortium. Avialable at: http://www.chess-project.org, accessed May, 2014.

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

[33] Timing Augmented Description Language (TADL2) syntax, semantics, metamodel Ver. 2, Deliverable 11, Aug. 2012.

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

[35] Rubus-ICE: Integrated Development Environment. http://www.arcticus-systems.com.

[36] AUTOSAR Techincal Overview, Release 4.1, Rev. 2, Ver. 1.1.0., The AUTOSAR Consortium, Oct., 2013. http://autosar.org.

[37] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Communications-Oriented Development of Component- Based Vehicular Distributed Real-Time Embedded Systems. *Journal of Systems Architecture*, 2013.

[38] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Support for end-to-end response-time and delay analysis in the industrial tool suite: Issues, experiences and a case study. *Computer Science and Information Systems, ISSN: 1361-1384*, 10(1), 2013.

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

[40] OMG Systems Modeling Language, version 1.3. http://www.omgsysml.-org.

[41] EAST-ADL Domain Model Specification, D 4.1.1. http://www.atesst.-org/home/liblocal/docs/ATESST2_D4.1.1_EAST-ADL2-Specification_-2010-06-02.pdf.

# II

# Included Papers

# Chapter 4

# Paper A: Communications-Oriented Development of Component-Based Vehicular Distributed Real-Time Embedded Systems

Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin
In Journal of Systems Architecture (JSA), vol 60, nr 2, Elsevier, 2014.

**Abstract**

We propose a novel model- and component-based technique to support communications-oriented development of software for vehicular distributed real-time embedded systems. The proposed technique supports modeling of legacy nodes and communication protocols by encapsulating and abstracting the internal implementation details and protocols. It also allows modeling and performing timing analysis of the applications that contain network traffic originating from outside of the system such as vehicle-to-vehicle, vehicle-to-infrastructure, and cloud-based applications. Furthermore, we present a method to extract end-to-end timing models to support end-to-end timing analysis. We also discuss and solve the issues involved during the extraction of these models. As a proof of concept, we implement our technique in the Rubus Component Model which is used for the development of software for vehicular embedded systems by several international companies. We also conduct an application-case study to validate our approach.

## 4.1 Introduction

In most of the model- and component-based software development strategies for automotive and other vehicular applications, models of the behavior of each on-board function are developed and successively refined to reach the implementation of each node or Electronic Control Unit (ECU). In this refinement process, the communications needed for each node are derived and a message set for each on-board network is defined. Moreover, timing parameters and requirements for each message are established. The majority of existing model- and component-based development approaches for vehicular distributed real-time embedded systems[1] allow for structural and functional modeling. They do not support *execution modeling* [1] which is concerned with the modeling of run-time properties and/or requirements (e.g., end-to-end deadlines and jitter) of software functions. The modeling of the 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 is very challenging because the functionality in the 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 challenges during the development of the systems in the industry is to model and express timing related information and perform timing analysis [2].

One way to deal with these challenges is to use a component technology that allows model- and component-based development of the 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 support the expression of timing related information and facilitate the identification of timing errors during the development by rendering the modeled application for end-to-end timing analysis with ease and unambiguity.

However, building such a component technology raises many challenges. One of the main reasons behind these challenges is that the development process for these 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

[1] Throughout the paper, we use the terms *system* or *application* to refer to component-based vehicular distributed real-time embedded system or application.

on specific platforms at a later stage. However, this way of development for the systems is often not practiced in the industry, especially in the automotive or vehicle domain. The traditional process for the development of these systems in the industry starts with designing the bus (or network) communication. The infrastructure for the system to be developed is already known. In the early stage of industrial development process, usually the focus is on finding the answers to the questions as follows. How many busses will there be 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, communications-oriented development process is used.

An important class of emerging distributed applications is novel functionality in road vehicles. These applications realize novel services based on vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communications. Both V2V and V2I are expected to support novel applications for road-safety, traffic efficiency, and driver/passenger comfort and entertainment. Already today, there are examples of traffic related cloud-services, e.g., community map and turn-by-turn navigation such as Waze[2] and traffic-congestion information by Google. However, to be successfully adopted, the development of these new applications needs to be integrated in contemporary workflow for development of vehicular functions. In most of the model- and component-based software development strategies, there is often non-existing, or limited, support to model network traffic originating from outside the vehicle. That is, traffic from V2V, V2I, and other, e.g., cloud-based applications are not naturally modeled and analyzed in existing approaches.

### 4.1.1 Goals and Paper Contributions

In order to provide a model- and component-based approach to support communications-oriented development of vehicular distributed real-time embedded systems, we target the following challenges in this paper.[3]

1. Modeling of legacy network communication.
   (a) Use of legacy (previously developed) nodes.
   (b) Development of new nodes that are deployed in legacy systems that use predefined communication rules.

[2] http://www.waze.com

[3] This work is an extension of our previous work [3].

(c) Adaptation[4] of a node when communication rules change without affecting its internal component design.

2. Extraction of end-to-end timing models from these systems.

3. Modeling and performing timing analysis of the applications that contain network traffic originating from outside of the system.

In order to provide proof of concept, we realize this technique in the existing industrial model the Rubus Component Model (RCM) [4]. We also conduct the automotive-application case study to validate our approach.

### 4.1.2 Paper Layout

The rest of the paper is organized as follows. Section 4.2 discusses the background and related work. In Section 4.3, we discuss the research problem in detail. In Section 4.4, we introduce a new approach for modeling legacy network communication. In Section 4.5, we discuss a method to extract end-to-end timing models. In Section 4.6, we present a case study. Section 4.7 concludes the paper and discusses the future work.

## 4.2 Background and Related Work

### 4.2.1 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[5] in close collaboration with several academic and industrial partners. Rubus is today mainly used for development of control functionality in vehicles by several international companies, e.g., BAE Systems Hägglunds[6], Volvo Construction Equipment[7], Knorr-bremse[8] and Mecel[9]. The Rubus concept is based around RCM and its development environment Rubus-ICE (Integrated Component development Environment) [5], which includes modeling

[4]We assume the adaptation (redeploying or upgrading) of a node is done offline. Dynamic adaptation and reconfiguration of the system is not within the scope of our current work.

[5]http://www.arcticus-systems.com

[6]http://www.baesystems.com/hagglunds

[7]http://www.volvoce.com

[8]http://www.knorr-bremse.com

[9]http://www.mecel.se

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, timing analyzable and synthesizable control functions in resource-constrained embedded systems.

The Rubus concept jointly considers the following viewpoints during the development. These viewpoints are also shown in Figure 4.1.

1. The viewpoint of the developer/designer.[10]
2. The viewpoint of the analysis framework.
3. The viewpoint of the run-time system.

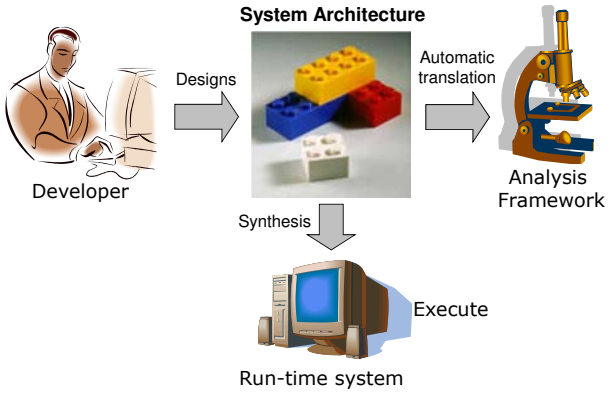


Figure 4.1: Three main viewpoints jointly considered in Rubus during the development.

In the viewpoint of the developer, the software architecture of the application is modeled in terms of software components and their interactions. This viewpoint consists of tools that handle software complexity, support appropriate level of expressiveness, and provide abstraction mechanisms that hide low-level details (such as source code).

In the viewpoint of the analysis framework, the software architecture is formal enough to render itself to automated analysis (e.g., response-time analysis). The analysis framework has the knowledge of the component architecture

[10]Developer refers to the application developer. We overload the terms “developer”, “designer” and “user” throughout the paper.

as well as the constraints and services provided by the run-time system. This viewpoint comprises of tasks (which are run-time entities), their activations, their interactions, and analysis models of tasks and messages. This viewpoint hides the complexity from the developer by providing automated analysis tools that extract the analysis models from the software architecture.

In the viewpoint of the run-time system, the synthesis takes (as input) the architecture design and possibly some artifacts (such as priorities for a task model) produced by the analysis framework, and maps it to the run-time system. The synthesis tools use the task model attributes and the component architecture, that is both syntactically and semantically correct, to generate code for the run-time system. This viewpoint provides sufficient run-time services to the components of the application while keeping a small footprint for the run-time system. With this view, the entire component framework is provided at development time, but only the parts that are used are mapped down to the actual run-time system.

#### The Rubus Component Model (RCM)

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. The control flow is expressed by triggering objects such as internal periodic clocks, interrupts, internal and external events. One important principle in RCM is to separate functional code and infrastructure implementing the execution model. The infrastructure is synthesized from the model.

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 trigger and data ports. Trigger and data correspond to trigger flow and data flow respectively. An SWC can be seen as a type, or a class, that can be instantiated an arbitrary number of times. By separating functional code from the infrastructure, RCM facilitates analysis and reuse of components in different contexts (an SWC has no knowledge how it connects to other components). Furthermore, the component model has a possibility to encapsulate SWCs into software assemblies enabling the designer to construct the system at different hierarchical levels.

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.

The software architecture of an example system modeled with RCM is depicted in Figure 4.2. The example shows how components interact with external events and actuators with regard to both data and triggering.

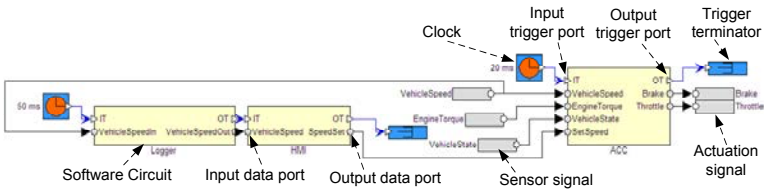


Figure 4.2: Example of the architecture of a system modeled in RCM.

### The Rubus Code Generator and Run-time System

Using the resulting software architecture of connected SWCs, the run-time system maps SWCs 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 other 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 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.

#### 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 (WCETs). 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 timing analysis supported by the model includes tighter response-time analysis of tasks with offsets [6], response-time analysis of Controller Area Network (CAN) [7, 8, 9], and distributed end-to-end response time and delay analysis [10].

### 4.2.2 Related Work

There are many modeling technologies that support component-based development of distributed systems, e.g., Distributed Component Object Model [11], Common Object Request Broker Architecture (CORBA) [12] and Enterprise JavaBeans (EJB) [13]. These models in their original form are not suitable for the development of resource-constrained distributed embedded systems with real-time requirements because they require excessive amount of computing resources, have large memory footprint and have inadequate support for modeling real-time communication. We focus on the component technologies that are targeted towards the vehicular domain.

#### AUTOSAR

AUTOSAR (AUTomotive Open System ARchitecture) [14] is an industrial initiative to provide standardized software architecture for the development of software in the automotive domain. 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). Furthermore, VFB hides the low-level implementation and communication details at the design time. We list some of the differences between AUTOSAR and RCM as follows.

- When AUTOSAR was being developed, there was no focus placed on its ability to specify and handle timing-related information such as real-time requirements and properties. On the other hand, these requirements and capabilities were taken into account right from the beginning during the development of the Rubus concept including RCM.

- AUTOSAR describes embedded software development at a higher level of abstraction compared to RCM. A Software Circuit in RCM more resembles to a runnable entity (schedulable element) compared to AUTOSAR SWC.

- Unlike AUTOSAR, RCM clearly distinguishes between the control flow and the data flow among SWCs within a node.

- In RCM, special network interface components 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- and inter-node communication at modeling level. There are no special components in AUTOSAR for modeling inter-node communication.

- AUTOSAR hides the modeling of the execution environment. On the other hand, RCM explicitly allows the modeling of execution requirements, e.g., jitter and deadlines, at an abstraction level close to the functional specification while abstracting the implementation details.

Despite these differences, there are some similarities between AUTOSAR and RCM, e.g., the sender receiver communication mechanism in AUTOSAR is very similar to the pipe-and-filter communication mechanism for components interconnection in RCM. In conclusion, AUTOSAR is more focussed on the functional and structural abstractions, hiding the implementation details about execution and communication. Whereas, RCM is all about modeling, analysis and synthesis of the execution environment of software functions. AUTOSAR hides the details that RCM highlights.

### TIMMO, TIMMO-2-USE, TADL and TADL2

TIMing MOdel (TIMMO) [2] is a large EU research project with both academic and industrial partners. It is more academic driven. It is an initiative to provide AUTOSAR with a timing model. The timing extensions proposed in this project are included in the version 4.0 of AUTOSAR specification [15]. TIMMO describes a predictable methodology and a language, Timing Augmented Description Language (TADL) [16], to express timing requirements and constraints during all design phases in the development of automotive embedded systems. TADL is inspired by Modeling and Analysis of Real Time and Embedded systems (MARTE) [17] 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 [18] which is a domain specific architecture description language targeted towards the automotive domain. TIMMO methodology and its model structure abstract the modeling of communication at implementation level of EAST-ADL where AUTOSAR is used. Hence, the modeling of intra- and inter-node communication mechanisms are the same as that of 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 the results of TIMMO project.

TIMMO-2-USE [19], another large EU research project, is a followup on TIMMO project. In this project, TADL2 language has been introduced which includes a major redefinition of TADL. TADL2 supports the AUTOSAR extensions regarding timing model. Apart from the redefinition of TADL, this project provides new algorithms, tools, and a methodology to model advanced timing information at different levels of abstraction. The use cases and validators indicate that the project results are in compliance with the AUTOSAR-based tool chain [15]. Since this project is recently finished, it may take some time for its results to become mature and find their way in the industrial use. Arcticus Systems has been involved in TIMMO-2-USE project as one of the industrial partners.

In conclusion, TIMMO methodology and TADL focus on expressing timing information. They are initiatives to annotate AUTOSAR with a timing model. This will be hard to accomplish all the way since AUTOSAR aims at hiding implementation details of execution environment and communication through the VFB. At the modeling level, there is no information in AUTOSAR to express low-level details, e.g., linking information. These details are necessary to extract the timing model from the architecture. There is no focus in this

initiative on how to extract this information from the model or perform timing analysis or synthesize the run-time framework. In our view, timing model means extracting enough information to be able to perform certain kind of timing analysis, e.g., end-to-end response-time analysis.

### ProCom

ProCom [20], developed as part of a research project at Mälardalen University, is a two-layered component model for the development of distributed embedded systems. At the upper layer, called ProSys, it models a system with concurrent subsystems that communicate with each other by means of asynchronous messages. At the lower layer, called ProSave, a subsystem is internally modeled in terms of functional components which are implemented as a piece of code, e.g., a C function. ProCom is inspired by RCM and there are a number of similarities between the ProSave modeling layer and RCM:

- components in both ProSave and RCM are passive,
- both models clearly separate data flow from control flow among their components,
- both models use pipe-and-filter style of communication mechanism for components interconnection.

However, ProCom does not differentiate between intra- and inter-node communication which is unlike RCM. ProCom hides communication details, whereas RCM lifts them up to the modeling level. It will be very hard in ProCom to extract the timing model and perform end-to-end timing analysis at the level where it is done in RCM.

### COMDES-II

COMDES-II [21], developed at the University of South Denmark, provides a component-based framework for the development of distributed embedded control systems. 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. Unlike RCM, COMDES-II employs signal-based communication for both intra- and inter-node interactions. COMDES-II does not include explicit components to

model network communication. Despite few differences, there are a number of similarities between RCM and COMDES-II. However, it will be very hard in COMDES-II to extract the timing model and perform end-to-end timing analysis at the level where it is done in RCM.

#### Middleware-based Approaches

Object Management Group defined middleware technologies such as Real-Time CORBA, minimum CORBA and CORBA lightweight services for the development of real-time and distributed embedded systems [22]. The run-time framework of Real-time CORBA is heavyweight. On the other hand, RCM has a small run-time footprint, i.e., timing and memory overhead. In RCM, we do the timing analysis and actually synthesize the application as run-time and communication platform efficient as possible. We believe, due to high resource requirements at run-time, Real-time CORBA is not efficiently usable in the type of applications that RCM focuses on.

There are other middleware solutions such as iLand project [23] in which a middleware-based framework is introduced to support predictable and time-bounded reconfiguration at run-time for the service-oriented distributed real-time systems. The methodology in this work supports composition of distributed applications based on the concept of services while taking into account real-time properties and requirements. This work focuses on service-oriented development, whereas our approach is based on component-based development. The framework in [23] relies on run-time mechanisms such as dual-band priority assignment [24] for dynamic resource management and reconfiguration. In [25], a component-based modeling approach is introduced that enables dynamic replacement of components at run-time while preserving the temporal properties of the system. On the other hand, we do not consider dynamic reconfiguration and replacement of components in our work. Most of these techniques have been validated for soft real-time systems in the multimedia-applications domain. However, our approach mostly focuses on hard real-time distributed embedded systems in the vehicular domain.

## 4.3 Problem Statement

To provide a model- and component-based approach to support communications-oriented development of the systems, we target the following issues.

### 4.3.1 Modeling of Legacy Network Communication

In an ideal scenario, it should be possible to automatically generate the communication from the design model for each distributed real-time application. However, this is often not the practice in the industry because of presence of legacy communications and legacy systems. These systems have their own predefined rules for communication. In order to support the modeling of these systems, the implementation details must be abstracted; the communication protocols must be encapsulated and abstracted; and adaptation of a node must be supported 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. This problem can be formulated as: *how to model legacy network communication and allow the use of legacy nodes to support the communications-oriented development processes for component-based distributed real-time embedded systems?*

### 4.3.2 Issues Concerning the Extraction of End-to-end Timing Model

In order to ensure that the 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. For this purpose, the end-to-end timing model should be unambiguously extracted from the modeled application. Moreover, the distributed transactions in the applications should be unambiguously identified, extracted and linked. The distributed transactions may consist of trigger chains, data chains or a combination of both. The first SWC in a trigger chain is triggered independently, while the rest of the SWCs are triggered by their respective predecessors as shown in Figure 4.3 (a). Whereas, each SWC in a data chain is triggered independently as shown in Figure 4.3 (b). A mixed chain is a combination of both trigger and data chains as shown in Figure 4.3 (c). The end-to-end timing model should include linking and mapping information of all these chains. The model should also identify the type of each chain because different timing constraints are specified on different types of chains [26]. Furthermore, data chains require different end-to-end timing analysis compared to trigger chains [10].

The linking and mapping problem is common in all types of chains. For simplicity, we consider the system which is modeled with only trigger chains as shown in Figure 4.4. 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

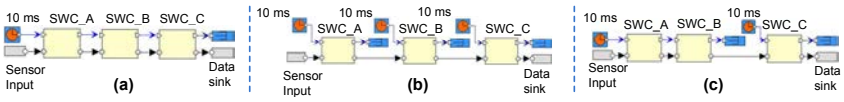


Figure 4.3: Example of (a) Trigger chain (b) Data chain (c) Mixed Chain.

using both inter- 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 trigger chain that is activated by a clock consists of four SWCs namely SWC1, SWC2, SWC4 and SWC5. We regard this chain as distributed trigger chain because it is distributed over more than one node. It is identified with the solid-line arrow in Figure 4.4. In this chain, 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[11]) and is received by SWC4 at the receiver node. SWC4 processes it and sends it to SWC5. The time elapsed between the event trigger at input of the task corresponding to SWC1 and production of the response of the task corresponding to SWC5 is referred to as the holistic or end-to-end response time of the distributed chain and is identified in Figure 4.4. The second distributed trigger chain that is activated by an external event consists of three SWCs namely SWC3, SWC6 and SWC7. It is identified by the dashed-line arrow in Figure 4.4.

There may not be direct triggering connections between any two neighboring SWCs in the chain which is distributed over more than one node, e.g., SWC2 and SWC4 in Figure 4.4. In this case, SWC2 communicates with SWC4 by sending signals over the network. Here, the problem is that when a trigger signal is produced by SWC2, it may not be sent straightaway as a message to the network. A message may combine several signals, and hence, there may be some waiting time for the signal to be sent to the network. The message may be sent periodically or sporadically or by any other rule defined by the underlying network protocol. When these trigger chains are modeled using the component-based approach, it is not straightforward to link 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 the chain: the $ID$ of the sender node; the $ID$ of the task corresponding to SWC that generated this message; the $ID$ of the destination node;

[11] We use the terms message and frame interchangeably because we only consider messages that fit into one frame.

and the $ID$ (s) of the task (s) corresponding to SWC (s) that should receive this message. In order to get a bounded end-to-end delay, a more important question is when and who triggers the destination SWC when a message is received at the destination node.

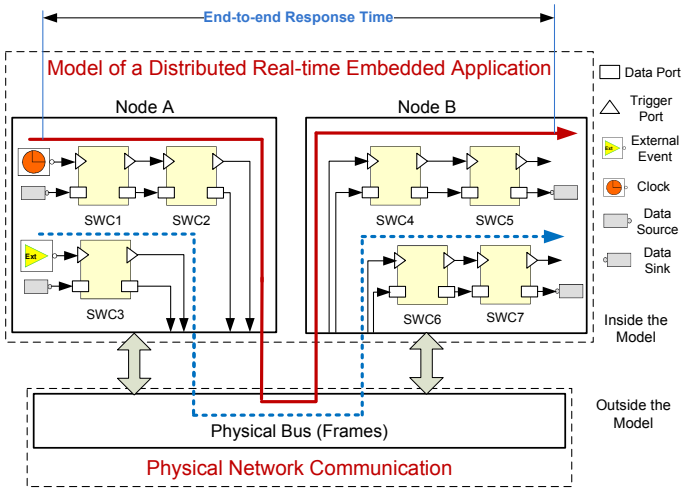


Figure 4.4: Example of distributed trigger chains.

The existing modeling components in RCM do not provide enough support to link and extract the corresponding timing information of distributed chains. Therefore, special objects in the component technology are needed to provide the linking information of distributed chains to extract end-to-end timing information. 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. 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 model of the network or any other model. The reason for the need to model exit and entry points for RCM models is to get the bounded delays for distributed chains. The model of entry and exit points will support the use of nodes developed using RCM with the nodes developed by other component technologies.

The problem discussed in this subsection can be formulated as: *how to*

*extract end-to-end timing models from component-based distributed real-time embedded systems that are built using the communications-oriented development processes?* We believe that the issues discussed in this subsection may occur during the development of any other component model for distributed real-time embedded systems that uses the pipe-and-filter communication mechanism for components interconnection, e.g., ProCom [20] and COMDES [21]. The problem of linking distributed 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.

### 4.3.3 Modeling and Timing Analysis of "Outside Traffic"

The problem arises when the requirements dictate the modeling and end-to-end timing analysis of the system at a stage where the models of some ECUs may not be available. However, the signals and messages which these missing ECUs are supposed to send and receive have been decided. In such a system, the network is assumed to contain "outside traffic", i.e., the messages whose sender nodes are not developed yet. This "outside traffic" could come from external services (e.g., V2V, V2I, and other cloud-based applications), from legacy nodes that lack proper behavior models, or from crude preliminary models of nodes that have not been completely modeled yet. Regardless of source, it is important to be able to analyze end-to-end timing behavior of vehicle internal functions, while taking into account the outside generated traffic. Similarly, the available ECUs may send messages via network to the nodes that will be available at a later stage. Some reasons behind these requirements are to support design space exploration, allow fine tuning of the system with respect to real-time requirements and detection of timing errors.

There exist timing dependencies among messages and their sender and receiver tasks. A message inherits some timing properties from its sender task, e.g., transmission type and period. If the sender task is triggered periodically then the message it sends is also periodic. Further, the message inherits period from its sender task. Similarly, if the sender task is activated sporadically then the corresponding message is sporadic and the message inherits *inhibit time* from the sender task. The inhibit time is the minimum amount of time that should elapse between two consecutive transmissions of a sporadic message. In the case of mixed transmission mode, the message inherits both period and inhibit time from its sender [9]. When the systems are analyzed, each message is assumed to inherit the release jitter from its sender (attribute inheritance [27]). For the messages whose sender tasks are unknown (because the sender

ECUs are not available yet or the network traffic is generated from outside of the model), these properties must be extracted in the timing model. Otherwise, the end-to-end timing analysis of these systems cannot be performed. The problem discussed in this subsection can be formulated as: *how to model and timing analyze the applications that contain network traffic originating from outside of the system?*

## 4.4 Modeling of Legacy Network Communication

We introduce a new modeling entity the Network Specification to represent the model of communication in a physical network. In order to abstract the implementation of communications in a node, we propose two special-purpose modeling entities namely Out and In Software Circuits for each frame that a node sends to and receives from the network respectively.

### 4.4.1 Network Specification (NS)

It is the model representation of a physical network. There is one NS for each network protocol. It consists of two parts, one is independent of the underlying communication protocol while the other is protocol dependent. The protocol-independent part defines messages and the data-elements mapped to them. A message is an entity that is used to send information from one node to another via network. Moreover, the protocol-independent part of the NS describes message properties such as a message ID, a unique sender node ID, a list of receiver nodes IDs and an ordered set of signals included in the message. For example, a signal in RCM has a name, data type, resolution and real-time properties. The protocol-independent part of NS also contains the list of nodes in the system.

The protocol-dependent part of the NS is uniquely defined for each protocol, e.g., it will be different for different higher-level protocols for Controller Area Network (CAN) [28] such as CANopen [29], Hägglunds Controller Area Network (HCAN) [30] and CAN for Military Land Systems domain (MilCAN) [31]. It defines the behavior semantics of each message according to the network communication protocol. It contains complete information of all frames which are sent to and received from the network. Moreover, it describes the frame properties. A frame is a formatted sequence of bits that is actually transmitted over the network. In RCM, a frame is a collection of RCM signals.

The frame properties described by the protocol-dependent part of the NS (e.g., for the CANopen protocol) include an identifier (a reference to the corre-

sponding message in the protocol-independent part), a priority, a transmission type (e.g., different types of message transmission in the CANopen protocol), a sender node ID, a list of receiver nodes IDs, whether the frame is an IN frame or an OUT frame, a period (period with which a message is sent in the case of periodic transmission), an inhibit time (minimum time between successive transmission of a message in the 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 (e.g., message deadline). Moreover, it also specifies the bus speed. The transmission type of a frame can be periodic, sporadic or mixed (transmitted periodically as well as sporadically) [9].

In RCM, the components inside a node communicate with each other via data and control signals. However, if a component on one node communicates with a component on another node via a network then the signals are packed into the frames. The frames are then transmitted over the network. Here, some questions arise concerning the communication in the network. How are signals mapped to messages? How are the signals packed into the frames? How are the signals encoded into the frames at the sender node? How are the signals decoded from the frames and sent to the respective SWCs at the receiver node? How many signals are there in each frame? All rules concerning the answers to these questions are specified in the Signal Mapping. The Signal Mapping is a unique object for each protocol for network communication and is an integral part of the protocol-dependent part of the NS. The Signal Mapping also describes the length of each signal in a frame, the type of signal encoding in a frame (e.g., signed or unsigned 2's complement), and maximum age of a signal guaranteed by the sender.

### 4.4.2 Out Software Circuit (OSWC)

It is the model representation of signals in an outgoing message to the network. Basically, it is a Software Circuit which denotes the data that leaves the model. There is one OSWC in a node for every outgoing frame on the network. Each OSWC describes the signals that can be sent in a particular frame. A frame contains zero or more signals. The 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 NS. Therefore, the number of data in-ports may vary depending upon the number of signals packed in the frame. The OSWC has no data and trigger out-ports. However, an optional trigger out-port can be used. It uses protocol-specific rules, specified in the protocol-specific part of the NS, while encoding data

and mapping signals to a frame. In this way, it 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 conceptual model of the OSWC is illustrated in Figure 4.5 (a), whereas its RCM model is shown in Figure 4.5 (b).

### 4.4.3 In Software Circuit (ISWC)

It is the model representation of signals in an incoming message from the network. Basically, it is a Software Circuit which denotes the data that enters the model. There is one ISWC component in a node for every frame received from the network. It describes all the signals that are contained in a received frame that is associated to it. The ISWC component has one trigger out-port that produces a trigger signal every time the component is executed. There is at least one data out-port in the ISWC. Each data out-port is associated with one signal in the NS. Therefore, the number of data out-ports may vary depending upon the number of signals contained in the received frame. There are no data in-ports in the ISWC. It has one trigger in-port which is triggered every time a frame arrives from the network. When a frame arrives at a node, the physical network drivers and protocol-specific implementation of the 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 linking and mapping information is provided in the NS), and the corresponding trigger port is triggered. Figures 4.5 (a) and 4.5 (a) graphically illustrate the conceptual and RCM models of the ISWC respectively. It should be noted that the developer can specify timing parameters such as execution times for the OSWC and ISWC.

The models of a node, a network, a signal database and a signal in RCM are shown in Figures 4.6 (a), (b), (c) and (d) respectively. The signal database object corresponds to the Signal Mapping which is part of the NS (as discussed in Section 4.4.1). It contains all the signals that are sent over the network. Each signal in the signal database is linked to one or more messages. The model of a message along with the list of user-defined properties is shown in Figure 4.6 (e). The developer specifies only the name, priority, data size, value and type of identifier (in the case of CAN) of the message. The message automatically inherits jitter, transmission type and period or inhibit time or both from the sender OSWC.

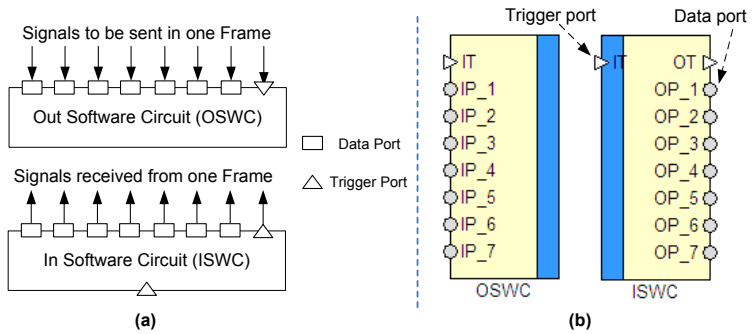


Figure 4.5: OSWC and ISWC components (a) conceptual models (b) models in RCM.

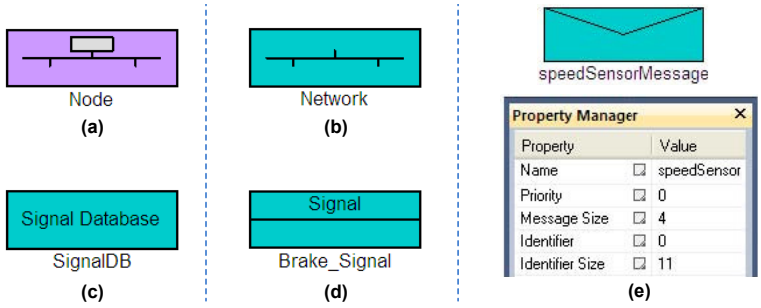


Figure 4.6: RCM models of (a) node, (b) network, (c) signal database, (d) signal, (e) message and its attributes.

Consider an example of a node in a distributed real-time embedded application modeled with the introduced objects as shown in Figure 4.7. Let the node be connected to the CAN network. The upper half of Figure 4.7 represents the model of a node, whereas the lower half represents the physical communication including the CAN controller and 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 grey 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 $R2$. All signals that are 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 NS. 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 physical transmission of this frame over the network according to the arbitration and communication rules specified by the CAN protocol.

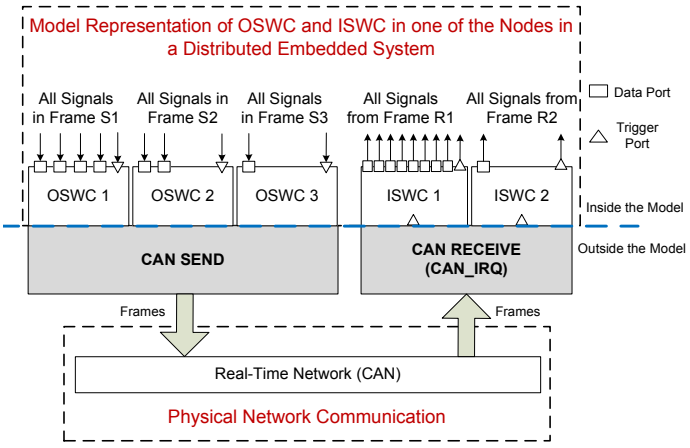


Figure 4.7: Model of a node using OSWCs and ISWCs for network communication.

When a frame arrives at the receiving node, it is transferred by the network drivers to the CAN RECEIVE grey box which is responsible for raising an interrupt request and passing the frame to the corresponding ISWC. In this case, the *TrigInterrupt* object in RCM corresponding to the interrupt is connected to the *in-port* of the ISWC. If CAN drivers use polling-based processing instead of interrupts then the *in-port* of the ISWC is connected to the clock, where clock period is set equal to the polling period. Upon receiving the frame, the ISWC 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 linking information available in the NS. It should be noted that when the OSWC 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 node to the network con-

troller or another model of communication network. Therefore, OSWC also represents the model of an exit point for RCM models. Similarly, ISWC component represents the model of an entry point for RCM models. Using our new approach, nodes can be developed without explicit knowledge about the communication configuration.

### 4.4.4 Automatic Generation of the OSWC and ISWC Components

Both OSWC and ISWC can be automatically generated from the NS by a Network Configuration Tool. The input to this tool is the protocol-specific information about the network communication and the linking information of tasks in all distributed chains (i.e., trigger, data and mixed) present in the application. This information is provided 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 the NS to the OSWC and ISWC and vice versa. One of the main purpose of our modeling technique is to use legacy components, however, it can also be used the other way around. That is, the protocol-independent part of the NS can be automatically generated from the models of the OSWC and ISWC from an existing system.

### 4.4.5 Support for Modeling "Outside Traffic"

A solution to the problem (discussed in Section 3.3) could be using the model of a dummy sender node in place of the ECU that is not available but decisions on the messages it sends are already taken. The only purpose of this node is to encode and pack signals into messages and send them to the network. Similarly, a dummy receiver node can be used to receive messages in place of a missing ECU. Such a solution can be realized in RCM with the models of sender and receiver nodes that contain one OSWC and one ISWC for every message they send and receive respectively. However, this solution is impractical as it adds design complexity to the system. Moreover, it adds an extra modeling and testing overhead on the developer because there are several modeling and specification steps involved when a node is modeled.

The problem can be solved in a better way by introducing a special type of message called stand-alone message. This message supports the modeling of "outside traffic". It does not bear any association with the OSWC component. This means, it does not have any sender task inside the model of the system.

However, there is an option for the user to associate this message to any number of ISWCs, i.e., it can be received by any number of tasks. Apart from the name, priority, data size, value and type of identifier (user-defined properties for a regular message); it is also possible for the user to specify the transmission type and corresponding timing parameters (period, inhibit time or both) for this message. The transmission type of a message is a very important parameter because the network timing analysis is dependent upon it especially in the case of CAN and its higher-level protocols. For example, if there are only periodic and sporadic messages in the system then one type of timing analysis is used such as [7]. On the other hand, if there is at least one mixed message in the system then a different timing analysis (i.e., response-time analysis for mixed messages) is used [9, 32, 33].

The user can also specify release jitter for the stand-alone message. The release jitter may either be equal to the difference between the estimated worst- and best-case response times of the sender (belonging to the node that is not available) or zero if these response times cannot be estimated at this stage. The extra user-defined information in the case of the stand-alone messages is vital for the network timing analysis, and hence, for the end-to-end timing analysis. The standalone message along with the list of its user-defined properties is shown in Figure 4.8. The dark vertical stripes on both of its sides differentiate it from the regular message in RCM. This message is treated differently from the regular message at the attribute inheritance step by the holistic timing analysis algorithm [27, 10].

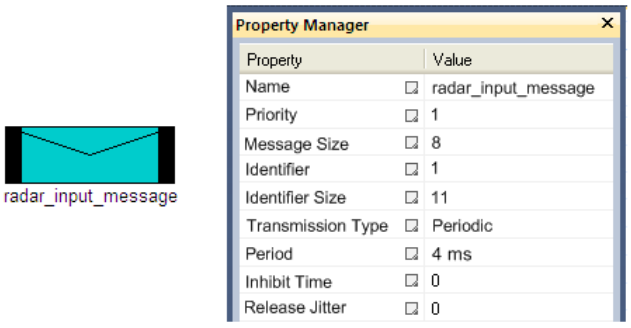


Figure 4.8: Model of a stand-alone message with the list of user-defined properties.

It should be noted that the list of user-defined properties of a stand-alone message (see Figure 4.8) is more general and includes user-defined properties of a regular message (see Figure 4.6 (e)). One may think of using the user-defined properties in Figure 4.8 consistently for all types of messages. However, this is not practical mainly because of two reasons. First, the timing information extracted from the modeled application may be redundant. That is, the transmission type and corresponding period and inhibit time will be extracted from the user-defined input as well as from the sender task. This redundancy may result in the extraction of ambiguous end-to-end timing model. Information duplication can lead to inconsistency in the model. Second, it will add extra complexity and burden on the developer to specify too much information during the modeling. Our intension is to extract unambiguous end-to-end timing information and keep things as simple as possible for the developer.

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

In order to ensure all timing requirements are met, the modeled application should render itself to the end-to-end timing analysis. For this purpose, the end-to-end timing model of the application should be available.

### 4.5.1 End-to-end Timing Model

This model consists of timing properties, requirements and dependencies concerning all tasks, messages, task chains and distributed transactions in the system under analysis. It consists of the following sub models:

1. System timing model
   (a) Node timing model
   (b) Network timing model
2. System linking model

#### System Timing Model

This model is composed of node and network timing models.

**1) Node timing model.** This model contains node-level timing information. We consider the model that is based on the transactional task model (i.e, tasks with offsets ) introduced by [34] and later on, extended by many researchers,

e.g., [35, 36]. 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 the events 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 $\Gamma_i$. Each task in $\Gamma_i$ 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.

- $C_{ij}$ denotes the worst-case execution time of the task.
- $O_{ij}$ denotes the offset of the task.
- $D_{ij}$ specifies the optional deadline of the task.
- $J_{ij}$ denotes the maximum release jitter.
- $B_{ij}$ represents the maximum blocking time which is the maximum time the task has to wait for a resource that is locked by a lower priority task. In order to obtain the blocking time for a task, a resource sharing protocol, e.g., Stack Resource Policy (SRP) [37] or Priority Ceiling Protocol (PCP) [38], that bounds the blocking time must be used.
- $P_{ij}$ denotes the priority of the task.
- $R_{ij}$ denotes the worst-case response time of the task.

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.

**2) Network timing model.** This model contains network-level timing information of the system. A network consists of a number of nodes that are connected through a real-time network. Currently, RCM supports CAN and its higher-level protocols such as CANopen, MilCAN and HCAN. However, it can be easily extended to support other protocols such as Flexray [39]. In this model, each message $m$ has the following attributes.

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

- $TRANSMISSION\_TYPE$ specifies whether the message is periodic or sporadic or mixed (both periodic and sporadic).
- $P_m$ denotes unique priority.
- $C_m$ specifies the transmission time.
- $J_m$ denotes the release jitter. Usually, it is inherited from the task that queues $m$.
- $s_m$ denotes the data payload in each message. It ranges from 0 to 8 bytes in a CAN message.
- $T_m$ specifies the period of a message in the case of periodic transmission. For a sporadic message, $MINT_m$ is used which refers to the minimum time that should elapse between the transmission of any two messages. For a mixed message, both $T_m$ and $MINT_m$ are specified.
- $B_m$ denotes the blocking time of the message. It refers to the maximum amount of time during which this message can be blocked by the lower priority messages.
- $R_m$ denotes the worst-case response time.

**System Linking Model**

In distributed embedded systems, there exist chains of components (tasks) that may be distributed over more than one node. A task chain consists of a number of tasks that are in a sequence and have one common ancestor. A task in a chain may receive trigger, data or both from its predecessor. Two neighboring tasks in a distributed transaction may reside on two different nodes, while the nodes communicate with each other via network. When there are chains in the system, the end-to-end timing model should not only contain timing related information but also the linking information among all tasks and messages within each distributed chain. All mapping and linking information of distributed chains is extracted into the system linking model.

### 4.5.2 Method to Unambiguously Extract and Link Distributed Chains

We provide a method to identify, extract and link distributed chains from the RCM models of component-based distributed real-time systems.

### Extraction of Unambiguous Timing Information

The end-to-end timing information that is extracted from all tasks, messages and distributed chains can be divided into two categories. The first category corresponds to the timing information that is provided by the user in the modeled application, e.g., most of the task and message attributes discussed in Section 4.5.1. Whereas, the second category corresponds to the timing information which is not directly provided by the user but has to be extracted from the modeled application. For example, release jitter for a message. It is inherited as the difference between the worst- and best-case response times of the sending task. Similarly, message transmission type, message period and inhibit times are often not specified by the user, rather they are inherited from the sender tasks. Hence, these parameters must be extracted from the modeled application and added in the timing model. We assign period or inhibit time to the message which is equal to the period or inhibit time of its sender task. If the sender task is activated by a clock, we assign periodic transmission type to the message. Similarly, if the sender task is activated by a sporadic event then we assign sporadic transmission type to the message. However, if the sender task is triggered by both a clock and a sporadic event then transmission type of the message is considered as mixed. These assignments are important because a message is analyzed differently based on its transmission type.

### Identification of Trigger, Data and Mixed Distributed Chains

In order to unambiguously identify each individual chain, we attach *trigger dependency* attribute with each task. This attribute is part of the data structure of tasks in the timing model. Basically, it extracts the triggering information for the corresponding task. If a task is triggered by an independent source, such as a clock, then this attribute is set to "independent". On the other hand, if the task is triggered by another task then this parameter is set to "dependent". A precedence constraint is also specified on this task in the case of dependent triggering.

An iterative method determines whether the triggering of every two neighboring tasks in a chain is dependent or independent of each other by testing the value of corresponding *trigger dependency* attributes. If this attribute for all tasks (except the first) is "dependent", the chain is identified as a trigger chain. On the other hand, if this attribute for each and every task in the chain has "independent" value, the chain is identified as a data chain. However, if this attributed has "independent" value for some tasks in the chain while "dependent" value for the rest, the chain is regarded as a mixed chain. This method is

applied to all chains in the system.

### Linking of Distributed Trigger, Data and Mixed Chains

The method for linking distributed chains is built upon the modeling approach that we discussed in the previous section. This method treats all types of chains in a similar fashion. The linking information for all distributed chains in the modeled application is provided in the NS. We assign references to trigger *in-ports* of OSWCs and the trigger *out-ports* of ISWCs along the same distributed chain. These references are contained in a reference array. There is one reference array for each distributed chain. The ordering of references within the array corresponds to the ordering of the components (OSWC/ISWC) along the trigger chain. That is, the first reference in the array corresponds to the trigger port of the first component in the chain, and so on. The reference arrays corresponding to all distributed chains in the system are specified in the NS. Consider the example shown in Figure 4.9. There are three SWCs in each node. The nodes are connected to the CAN network. There are two trigger chains in the system that are distributed over two nodes. These chains are identified as $TC_1$ and $TC_2$ as shown below.

- $TC_1 : SWC1 \rightarrow SWC2 \rightarrow OSWC\_A1 \rightarrow ISWC\_B1 \rightarrow SWC4 \rightarrow SWC5$.
- $TC_2 : SWC6 \rightarrow OSWC\_B1 \rightarrow ISWC\_A1 \rightarrow SWC3$.

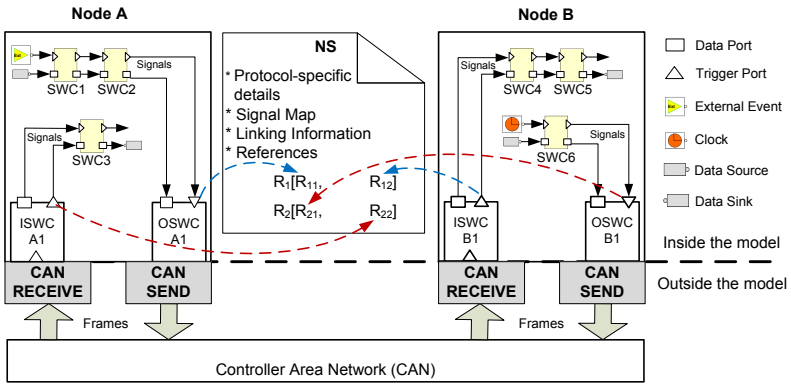


Figure 4.9: Demonstration of linking the distributed chains.

The trigger chain $TC_1$ is triggered by an external event, whereas $TC_2$ is triggered by a clock. There is a reference array $R_1$ that contains references to all the OSWC and ISWC components in $TC_1$. $R_{11}$ refers to the trigger *in-port* of OSWC A1 in Node A, whereas $R_{12}$ refers to the trigger *out-port* of ISWC B1 in node B. Similarly, a reference array $R_2$ is stored in the NS that contains references to all OSWCs and ISWCs in $TC_2$. $R_{21}$ refers to the trigger *in-port* of OSWC B1 in Node B, whereas $R_{22}$ refers to the trigger *out-port* of ISWC A1 in node A. In this way, all the neighboring components located in different nodes within a distributed trigger chain can be linked.

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

In Rubus-ICE, the application is modeled in the Rubus Designer tool. It is then compiled to the Intermediate Compiled Component Model (ICCM). Apart from the compiled component model, the ICCM file also includes timing and linking information of the modeled system. The timing model that is implemented in the Rubus Analysis Framework, extracts the required timing and linking information from the ICCM file format[12] as shown in Figure 4.10. From the extracted 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 and delays of distributed chains, network utilization, etc., back to the Rubus-ICE tool suite.

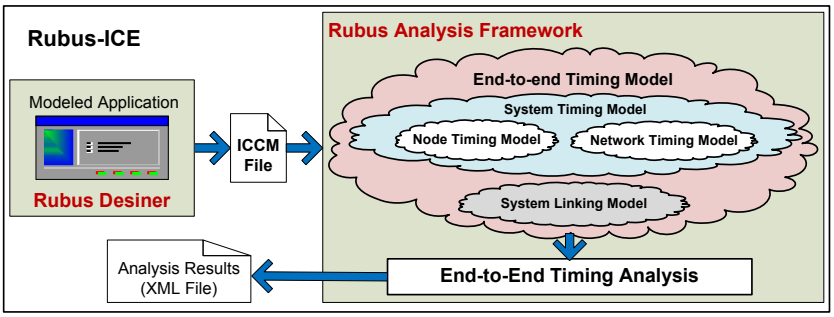


Figure 4.10: Extraction of the end-to-end timing model in Rubus-ICE tool-suite.

[12] in XML format.

## 4.6 Automotive-application Case Study

We provide a proof of concept for the modeling technique and timing model extraction method that we implemented in Rubus-ICE by conducting an automotive-application case study. We model the next-generation Adaptive Cruise Control system with RCM and analyze it with the Holistic Response Time Analysis (HRTA) plug-in in Rubus-ICE.

### 4.6.1 Next-generation Adaptive Cruise Control System

The Adaptive Cruise Control (ACC) system is an automotive feature that allows a vehicle to automatically adapt itself to the traffic environment to maintain a steady speed to the value that is preset by the driver. Often, it uses a radar to create a feedback of distance to, and velocity of, the preceding vehicle. It also communicates (cooperates) with the surrounding vehicles. Moreover, it receives traffic related cloud-services such as community map and turn-by-turn navigation services from outside of the 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. The ACC system may be modeled with four nodes namely Cruise Control (CC), Engine Control (EC), Brake Control (BC) and User Interface (UI) [40]. Figure 4.11 shows the block diagram of the ACC system. The nodes communicate with each other via CAN network.

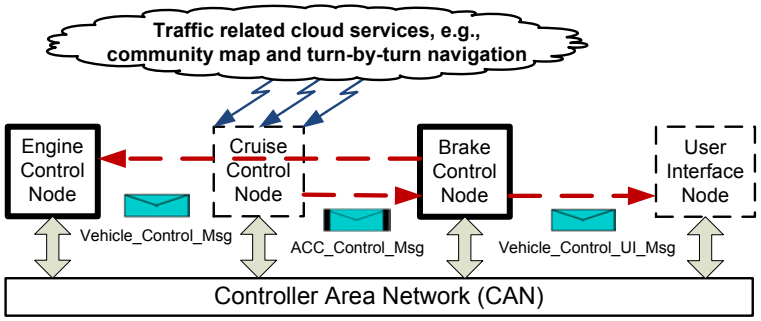


Figure 4.11: Block diagram of Adaptive Cruise Control System.

Assume that the models of EC and BC nodes are available while the models of CC and UI nodes will be available at a later stage. However, the decisions

about network communication have been made. There is one stand-alone message "ACC_Control_Msg" in the system that is assumed to be sent by the CC node (not available yet). This message is received by the BC node as shown by the dashed-line arrow in Figure 4.11. The BC node sends two messages over the network. First message "Vehicle_Control_UI_Msg" is sent to the UI node (not available yet). Whereas, the second message "Vehicle_Control_Msg" is sent to the EC node. It should be noted that the dashed-line arrows represent virtual communication while the actual message transmission takes place through the CAN network.

The UI node reads driver inputs 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 that corresponds to ON/OFF, Standby and Resume states for ACC; Set Speed input (desired cruising speed set by the driver) and desired clearing distance from the preceding vehicle. This node receives linear and angular speed, status of manual brake sensor, and status messages and warnings to be displayed on the screen from the BC node via the CAN network.

The CC node analyzes the state of the cruise control switch. If the switch is in the ON state then the cruise control functionality is activated. It reads input from a proximity sensor (e.g., radar) and processes it to determine the presence of the vehicle in front of it. It also receives V2V communication and navigation information from outside of the vehicle as shown in Figure 4.11. Moreover, it processes the radar signals along with the other information, such as vehicle speed, to determine its distance to the preceding vehicle. It sends a CAN message to the BC node. The message carries the control information that is used to adjust the speed of the vehicle with respect to the cruising speed or clearing distance from the preceding vehicle.

The EC node is responsible for controlling vehicle speed by adjusting the engine throttle. It reads sensor input and determines engine torque. It receives a CAN message (from the BC node) that includes information regarding vehicle speed and status of the manual brake sensor. 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 the engine throttle.

The BC node receives signals from the break sensors. It also receives the status from the linear and angular speed sensors which are connected to the wheels. It receives a CAN message that includes control information processed by the CC node. Based on this feedback, it computes new vehicle speed. It produces control signals and sends them to the brake actuator and brake light controller. It also sends CAN messages to the EC and UI nodes that carry

information regarding status of manual brake, vehicle speed and angular speed.

### 4.6.2 Modeling of the ACC System in Rubus-ICE

The RCM model of ACC system is shown in Figure 4.12. Since, CC and UI nodes are not available at this stage, there are models of only CC and EC nodes in the application. The model of CAN bus is also shown. The selected speed of CAN bus is 500 kbps. The standard CAN frame format is selected.

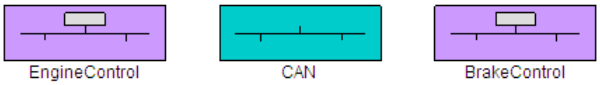


Figure 4.12: Adaptive Cruise Control System modeled with RCM.

There are three CAN messages in the system ACC_control_Msg, Vehicle_Control_Msg and Vehicle_Con-trol_UI_Msg as shown in Figure 4.13. ACC_control_Msg is the only stand-alone message. The senders and receivers of all messages are shown in Figure 4.11. A signal database is also shown in Figure 4.13. It corresponds to the NS (see Section 4) and contains all the signals that are sent over the network. Each signal in the signal database is linked to one or more messages. The user-defined properties of all messages are also visible in Figure 4.13.

The internal architecture of the BC node is shown in Figure 4.14. It is modeled with five SWCs ( SpeedSensorInput, ManualBrakeSensorInput, RMPSensorInput, SetBrakeSignal_SWC and SetBrakeLightSignal_SWC), one ISWC component (ACC_control_Msg_ISWC), two OSWC components (Vehicle_Control_Msg_OSWC and Vehicle_Control_UI_Msg_OSWC) and one assembly denoted by Brake_Control. An assembly in RCM is a container for various software items. The Brake_Control assembly is further modeled with two SWCs BrakeInputInfoProcessing and BrakeController as shown in Figure 4.15. Each component is named according to its functional behavior, e.g., the ACC_control_Msg_ISWC component is responsible for receiving ACC_control_Msg from the network.

The internal architecture of EC node is shown in Figure 4.16. It is modeled with two SWCs (EngineTorqueInput and SetThrottlePosition), one ISWC component (Vehicle_Con-trol_Msg_ISWC) and one assembly (Engine_Control) as shown in Figure 4.17. The Engine_Control assembly is further modeled with two SWCs EngineInputInformationProcessing and ThrottleControl.

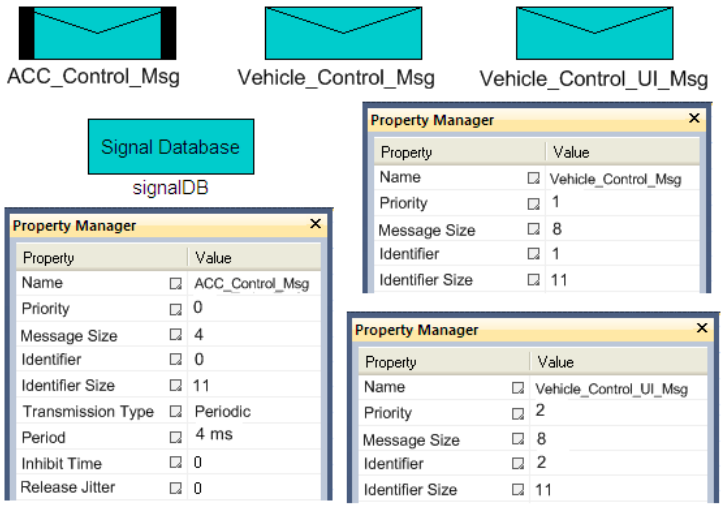


Figure 4.13: CAN messages and signal database modeled with RCM.

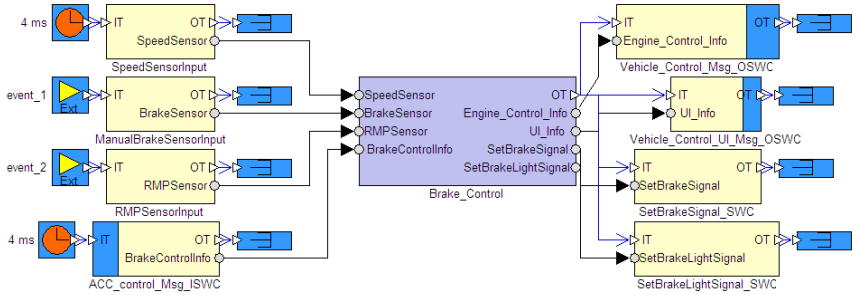


Figure 4.14: RCM model of the Brake Control node.

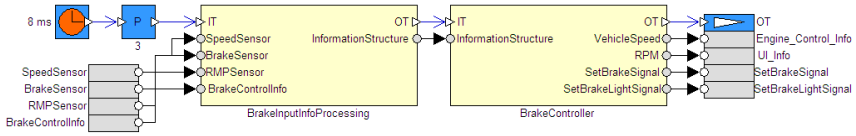


Figure 4.15: Internal model of Brake_Control assembly in RCM.

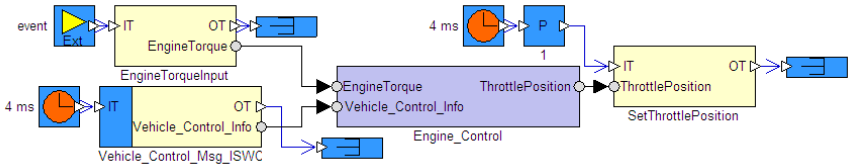


Figure 4.16: RCM model of the Engine Control node.

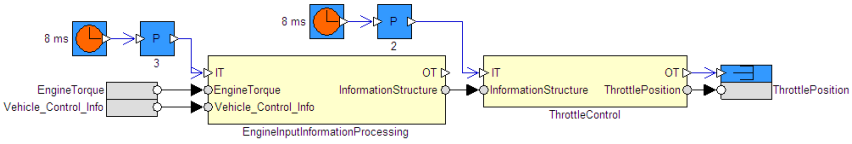


Figure 4.17: Internal model of Engine_Control assembly in RCM.

### 4.6.3 Holistic Response-time Analysis of the ACC System

The HRTA plug-in in Rubus-ICE calculates the response times of all messages and tasks as well as end-to-end or holistic response times of Distributed Transactions (DTs). We refer the reader to [41] for the details about the holistic response-time analysis. We focus on the analysis of the following DTs.

1. $DT_1$: ACC_control_Msg → ACC_control_Msg_ISWC → BrakeInputInfoProcessing → BrakeController → SetBrakeSignal_SWC
2. $DT_2$: ACC_control_Msg → ACC_control_Msg_ISWC → BrakeInputInfoProcessing → BrakeController → SetBrakeLightSignal_SWC
3. $DT_3$: SpeedSensorInput → BrakeInputInfoProcessing → BrakeController → Vehicle_Control_UI_Msg_OSWC → Vehicle_Control_UI_Msg
4. $DT_4$: SpeedSensorInput → BrakeInputInfoProcessing → BrakeController → Vehicle_Control_Msg_OSWC → Vehicle_Control_Msg → Vehicle_Control_Msg_ISWC → EngineInputInformationProcessing → ThrottleControl → SetThrottlePosition

Both distributed transactions $DT_1$ and $DT_2$ are initiated by the stand-alone message ACC_control_Msg. They terminate by producing the control signals for brake actuators and brake light controllers. $DT_3$ starts with the speed sensor input in the BC node and terminates by sending the message destined for the

Table 4.1: Calculated holistic response times of distributed transactions under analysis.

| **Distributed Transaction** | $DT_1$ | $DT_2$ | $DT_3$ | $DT_4$ |
|---|---|---|---|---|
| **Holistic Response Time ($\mu$s)** | 520 | 555 | 1250 | 830 |

UI node. Finally, $DT_4$ initiates with the speed sensor input in the BC node and terminates by producing a control signal for engine throttle controller in the EC node. The worst-case execution times of all SWCs are selected in the range of $(20 - 200)\mu s$. The holistic response times of these distributed transactions are shown in Table 9.3. In order to interpret the calculated results, consider the holistic response time of $DT_4$ in Table 9.3. It indicates that the maximum time required from sensing the variation in the vehicle speed to controlling the engine throttle actuator is $830\mu s$. The holistic response times of other DTs can be interpreted in a similar fashion.

## 4.7 Conclusion and Future Work

We introduced a new technique to provide a model- and component-based support for communications-oriented development of vehicular distributed real-time embedded systems. The proposed approach allows modeling of legacy network communication and abstracts the implementation and configuration of communications in the component-based systems. It explicitly enables the communication capabilities of a node, but hides the implementation or protocol details. Moreover, it 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 architecture. In order to support end-to-end timing analysis, we presented a method to extract end-to-end timing models from the systems that are developed using the proposed approach. In this context, we discussed and resolved various issues. Our technique also supports modeling and performing timing analysis of distributed applications that contain network traffic originating from outside of the system, e.g., cloud-based applications. As a proof of concept, we implemented this technique in the existing industrial tool Rubus-ICE, and validated it by modeling and analyzing the automotive application-case study. We believe,

this technique may be suitable for several other model- and component-based development technologies that use a pipe-and-filter style for component interconnection, e.g., ProCom and COMDES. Moreover, it 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. We believe, the tools implementing our technique may prove helpful for the software development organizations in the vehicular domain to decrease the costs for software development, configuration and testing.

An interesting future research direction is to bridge the semantic gap between functional models (expressed in standard languages as EAST-ADL and/or proprietary languages such as Simulink or Statemate) and execution models (expressed in proprietary languages like RCM). It would also be interesting and useful to facilitate the exchange of timing analysis models and tools between RCM and several other component models and tools.

## Acknowledgement

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

# Bibliography


[1] Jukka Mäki-Turja, Kaj Hänninen, and Mikael Nolin. Towards efficient development of embedded real-time systems, the component based approach. In *International Conference on Embedded Systems & Applications (ESA), 2006*, June 2006.

[2] TIMMO Methodology, Version 2, Deliverable 7, Oct. 2009.

[3] Saad Mubeen, Mikael Sjödin, and Jukka Mäki-Turja. Supporting early modeling and end-to-end timing analysis of vehicular distributed real-time applications. In *Real-Time and Distributed Computing in Emerging Applications (REACTION) Workshop located at 33rd IEEE Real-Time Systems Symposium (RTSS), 2012*, December 2012.

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

[5] Rubus-ICE: Integrated component Development Environment, 2013. http://www.arcticus-systems.com.

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

[7] 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.

[8] 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.

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

[10] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Support for end-to-end response-time and delay analysis in the industrial tool suite: Issues, experiences and a case study. In *Computer Science and Information Systems, vol. 10, no. 1, pp 453-482, January 2013. ISSN: 1361-1384*.

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

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

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

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

[15] Mastering Timing Information for Advanced Automotive Systems Engineering. In the TIMMO-2-USE Brochure, 2012. Available at: http://www.timmo-2-use.org/pdf/T2UBrochure.pdf. 2012.

[16] TADL: Timing Augmented Description Language, Version 2, Deliverable 6, October 2009. The TIMMO Consortium.

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

[18] EAST-ADL Domain Model Specification, Deliverable D4.1.1, 2010. http://www.atesst.org/home/liblocal/docs/ATESST2_D4.1.1_EAST-ADL2-Specification_2010-06-02.pdf.

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

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

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

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

[23] M. Garcia Valls, I.R. Lopez, and L.F. Villar. iland: An enhanced middleware for real-time reconfiguration of service oriented distributed real-time systems. *IEEE Transactions on Industrial Informatics*, 9(1):228–236, 2013.

[24] M. Garcia Valls, Alejandro Alonso, and Juan Antonio de la Puente. A dual-band priority assignment algorithm for dynamic qos resource management. *Future Generation Computer Systems*, 28(6):902–912, 2012.

[25] J. C. Romero and M. Garcia Valls. Scheduling component replacement for timely execution in dynamic systems. *Software- Practice and Experience*, 2013.

[26] 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 *Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS), 2008*, dec. 2008.

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

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

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

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

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

[32] S. Mubeen, J. Mäki-Turja and M. Sjödin. Response-Time Analysis of Mixed Messages in Controller Area Network with Priority- and FIFO-Queued Nodes. In *9th IEEE International Workshop on Factory Communication Systems (WFCS), 2012*, May 2012.

[33] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Worst-case response-time analysis for mixed messages with offsets in controller area network. In *17th IEEE Conference on Emerging Technologies and Factory Automation (ETFA), 2012*, sept. 2012.

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

[35] 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.

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

[37] T. P. Baker. Stack-based scheduling for realtime processes. *Real-Time Systems*, 3(1):67–99, April 1991.

[38] L. Sha, R. Rajkumar, and J.P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, 1990.

[39] FlexRay Consortium, FlexRay Communications System - Protocol Specication Version 2.1 Revision A, December 2005. http://www.flexray.com/.

[40] Adaptive Cruise Control System Overview. In *5th Meeting of the U.S. Software System Safety Working Group. Available: http://sunnyday.mit.-edu/safety-club/workshop5/Adaptive_Cruise_Control_Sys_Overview.pdf*, April 2005.

[41] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Support for Holistic Response-time Analysis in an Industrial Tool Suite: Implementation Issues, Experiences and a Case Study. In *19th IEEE Conference on Engineering of Computer Based Systems, 2012*, pages 210 –221, April 2012.

# Chapter 5

# Paper B: Integrating Mixed Transmission and Practical Limitations with the Worst-Case Response-Time Analysis for Controller Area Network

Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin
Conditionally accepted to the Journal of Systems and Software (JSS), Elsevier, April, 2014.

**Abstract**

The existing worst-case response-time analysis for Controller Area Network (CAN) calculates upper bounds on the response times of messages that are queued for transmission either periodically or sporadically. However, it does not support the analysis of mixed messages. These messages do not exhibit a periodic activation pattern and can be queued for transmission both periodically and sporadically. They are implemented by several higher-level protocols based on CAN that are used in the automotive industry. We extend the existing analysis to support worst-case response-time calculations for periodic and sporadic as well as mixed messages. Moreover, we integrate the effect of hardware and software limitations in the CAN controllers and device drivers such as abortable and non-abortable transmit buffers with the extended analysis. The extended analysis is applicable to any higher-level protocol for CAN that uses periodic, sporadic and mixed transmission modes.

## 5.1 Extended Version

This paper extends our previous works that are published in the conferences as a full paper in the IEEE conference on Emerging Technologies and Factory Automation (ETFA-2011) [1] and two work-in-progress papers (discussing basic ideas and preliminary work) in the IEEE Symposium on Industrial Embedded Systems (SIES-2012) [2] and ETFA-2012 [3] respectively. To be precise, the work in this paper generalizes the response-time analysis for Controller Area Network (CAN) developed in [1] by extending the proposed analyses in [2] and [3]. In addition, we conduct a case study to show a detailed comparative evaluation of the extended analyses.

## 5.2 Introduction

Controller Area Network (CAN) [4] is a multi-master, event-triggered, serial communication bus protocol supporting bus speeds of up to 1 mega bits per second. It has been standardized by the International Organization for Standardization as ISO 11898-1 [5]. It is a widely used protocol in the automotive domain. According to CAN in Automation (CiA) [6], the estimated number of CAN enabled controllers sold in 2011 are about 850 million. In total, more than two billion CAN controllers have been sold until 2011. Out of this huge number, approximately 80% CAN controllers have been used in the automotive domain [7]. CAN also finds its applications in other domains, e.g., industrial control, medical equipments, maritime electronics, and production machinery [8]. There are several higher-level protocols for CAN that are developed for many industrial applications such as CAN Application Layer (CAL), CANopen, J1939, Hägglunds Controller Area Network (HCAN), CAN for Military Land Systems domain (MilCAN).

CAN is often used in hard real-time systems that have stringent deadlines on the production of their responses. The providers of these systems are required to ensure that the systems meet their deadlines. Moreover, the need for safety criticality in most of these 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. For this purpose, *a priori* analysis techniques such as schedulability analysis [9, 10, 11, 12, 13] have been developed by the research community.

Response Time Analysis (RTA) [14, 9, 10, 11, 12] is a powerful, mature and well established schedulability analysis technique. It is a method to cal-

culate upper bounds on the response times of tasks or messages in a real-time system or a 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) [15].

### 5.2.1 Motivation and Related Work

RTA for CAN was developed by Tindell et al. [16] 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, e.g., Volcano Network Architect (VNA) [17]. Furthermore, this analysis has served as the basis for many research projects. Later on, Davis et al. [18] refuted, revisited and revised the analysis developed by Tindell et al. The revised analysis is also implemented in the existing industrial tool suite Rubus-ICE [19, 20] which is used by several international companies.

The analysis in [16, 18] assumes that each node picks up the highest priority message (that is ready for transmission) from its transmit buffers when entering into the bus arbitration. This assumption may not hold in some cases due to different types of queueing policies and hardware limitations in the CAN controllers [8, 21, 22]. The different types of queueing polices implemented by the CAN device drivers and communications stacks, internal organization, and hardware limitations in CAN controllers may have significant impact on the timing behavior of CAN messages. Various practical issues and limitations due to deviation from the assumptions made in the seminal worst-case analysis for CAN are discussed in [23] and analyzed by means of message traces in [8]. These limitations are not considered in the seminal RTA for CAN by Tindel et al. [16] and revised RTA for CAN by Davis et al. [18]. A few examples of these limitations that are considered in RTA for CAN are controllers implementing First-In, First-Out (FIFO) and work-conserving queues [24, 25, 22], limited number of transmit buffers [26, 27], copying delays in transmit buffers [28], transmit buffers supporting abort requests [21], the device drivers lacking abort request mechanisms in transmit buffers [28], and protocol stack prohibiting transmission abort requests in some configurations as in the case of AUTOSAR [29].

Let us first consider the related works on RTA for CAN with different queueing policies. The analysis in [16, 18] assumes that all CAN device drivers

implement priority-based queues. In [25, 22] Davis et al. pointed out that this assumption may become invalid when some nodes in the network implement FIFO queues. Some examples of the CAN controllers implementing FIFO queues are Infineon XC161CS, Microchip PIC32MX, Renesas R32C/160 and XILINX LogiCORE IP AXI Controller [22, 21]. Davis et al. [25, 22] extended the analysis for CAN where some nodes implement priority queues while others implement FIFO. In the works in [25, 22], the message deadlines are assumed to be smaller than or equal to the corresponding periods. In [24], Davis et al. lifted this assumption by supporting the analysis for CAN messages with arbitrary deadlines. Furthermore, they extended their previous works [25, 22] to support RTA for CAN with FIFO and work-conserving queues.

Now we discuss the related works on RTA for CAN with practical limitations in transmit message buffers of the CAN controllers. The analysis in [16, 18] assumes that the CAN controllers have large number of transmit buffers. However, some CAN controllers have small number of transmit buffers, e.g., the CAN controllers 8xC592, SJA1000 and 82C200 by Philips have less than three transmit buffers [22, 28, 24]. If a CAN controller has less than three transmit buffers and does not support transmission abort requests as in the case of Philips 82C200, a higher priority message released in the same controller may suffer from priority inversion [27, 16, 28]. That is, if all buffers in the CAN controller are occupied by lower priority messages, a higher priority message released in the same controller has to wait for one of the lower priority messages to transmit thereby vacating a space in the transmit buffer. During this waiting time, priority inversion occurs that adds an additional delay to the response time of the higher priority message.

The priority inversion can occur even if the controllers support transmission abort requests [26, 21]. Consider the case in which there are two transmit buffers in every CAN controller. If a higher priority message becomes ready when both transmit buffers are occupied by the lower priority messages, the lowest priority message in the transmit buffer (that is not under transmission) is swapped with the higher priority message from the message queue. During the swapping process, it may be possible that the lower priority message from the second buffer finishes its transmission and the next arbitration period starts. At this point, both buffers may be empty while any other lower priority message from another node wins the arbitration and starts to transmit. This causes priority inversion for the higher priority message that is being swapped. In [26], Meschi et al. proved that this type of priority inversion can be avoided if the controller implements at least three transmit buffers. According to [21], most of the CAN controllers implement more than three

transmit buffers and are capable of aborting transmission requests, e.g., Atmel AT89C51CC03/AT90CAN32/64 and Microchip MPC2515 [22].

The analysis in [26] does not account the overhead of the copying delay. Khan et al. [21] integrated this extra delay with the analysis in [16, 18] caused by priority inversion due to transmission abort requests supported in the CAN controllers. In the case of CAN controllers implementing non-abortable transmit requests (with more than three transmit buffers), RTA for CAN is extended in [28, 27].

In order to avoid deadlines violations due to high transient loads, current automotive embedded systems are often scheduled with offsets, i.e., using externally imposed delays between the times when the messages can be queued [30]. Furthermore, the worst-case response times of messages (especially with lower priority) in CAN increase with the increase in the network load. However, the worst-case response-times of lower priority messages in CAN can be reduced if the messages are scheduled with offsets [31, 32, 33]. A method for the assignment of offsets to improve the overall bandwidth utilization is proposed in [32, 33]. The worst-case RTA for CAN messages that are scheduled with offsets has been developed by several researchers [34, 35, 31, 36, 30].

However, all these analyses assume that the messages are queued for transmission either periodically or sporadically. They do not support mixed messages in CAN, i.e., the messages that are simultaneously time (periodic) and event (sporadic) triggered. Mixed messages are implemented by several higher-level protocols based on CAN that are used in the automotive industry. Mubeen et al. [1] extended the seminal analysis [16, 18] to support mixed messages in CAN where nodes implement priority-based queues. Mubeen et al. [37] further extended their analysis to support mixed messages in the network where some nodes implement priority queues while others implement FIFO queues. Mubeen et al. also extended the existing analysis for CAN to support worst-case response-time calculations for periodic, sporadic and mixed messages that are scheduled with offsets [38, 39]. In [2] and [3] we presented the basic idea and work in progress on the RTA of mixed messages in CAN with controllers implementing abortable and non-abortable transmit buffers respectively.

### 5.2.2 Paper Contributions

In this paper, we extend and generalize the RTA for periodic, sporadic and mixed messages in CAN by integrating it with the effect of buffer limitations in the CAN controllers namely abortable and non-abortable transmit buffers. The relationship between the existing and extended RTA for CAN is shown

in Figure 5.1. The analyses enclosed within the dashed-line box in Figure 5.1 are the focus of this paper. The extended analysis is able to analyze network communications in not only homogeneous systems, but also heterogeneous systems where:

1. CAN-enabled Electronic Control Units (ECUs) are supplied by different tier-1 suppliers such that some of them implement abortable transmit buffers, some implement non-abortable transmit buffers, while others may not have buffer limitations because they implement very large but finite number of transmit buffers.

2. any higher-level protocol based on CAN is employed that uses periodic, sporadic and mixed transmission modes for messages.

It should be noted that the main contribution in this paper, compared to the contributions in [1, 2, 3], is that the extended analysis is also applicable to the heterogeneous systems. Moreover, we conduct a case study to show the applicability of the extended analyses. We also carry out a detailed comparative evaluation of the extended analyses.

### 5.2.3 Paper Layout

The remainder of the paper is organized as follows. In Section 5.3, we discuss mixed transmission modes implemented by higher-level protocols based on CAN. Section 5.4 describes the system model. In Section 5.5, we present the extended RTA for mixed messages without buffer limitations. Sections 5.6 and 5.7 discuss the extended RTA for mixed messages in the case of abortable and non-abortable transmit buffers respectively. Section 5.8 discusses the case study and presents the comparative evaluation of the extended analyses. Section 5.9 concludes the paper and discusses the future work.

## 5.3 Mixed Transmission Patterns Implemented by Higher-level Protocols

When CAN is employed for network communication in a distributed real-time system, each node (processor or ECU) is equipped with a CAN interface that connects the node to the network [40]. Application tasks in each node, that require remote transmission, are assumed to queue messages for transmission over the CAN network. The messages are transmitted according to the protocol

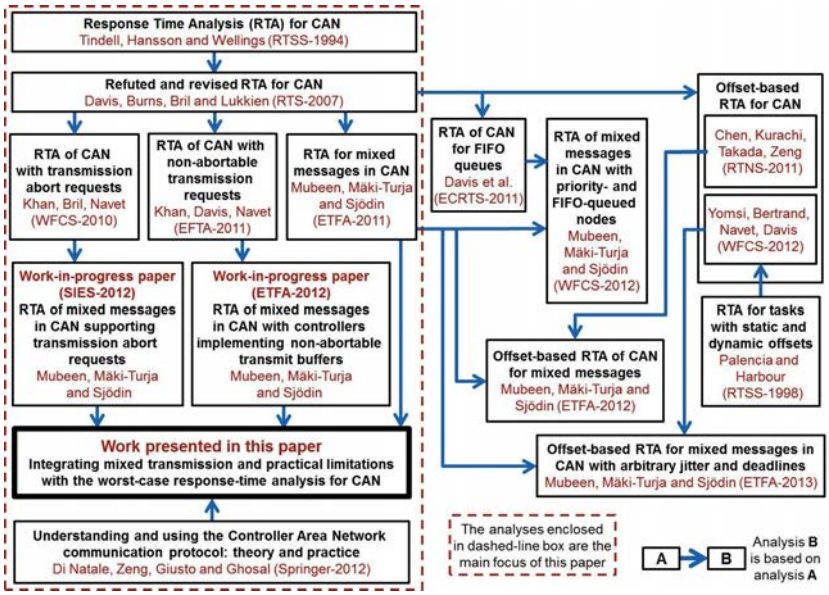


Figure 5.1: Relation between the existing and extended response-time analyses for CAN

specification of the CAN protocol. Traditionally, it is assumed 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 some higher-level protocols and commercial extensions of CAN in which the task that queues the messages can be invoked periodically as well as sporadically. If a message can be queued for transmission periodically as well as at the arrival of a sporadic event then the transmission type of the message is said to be mixed. In other words, a mixed message is simultaneously time (periodic) and event triggered (sporadic). We identified three types of implementations of mixed messages used in the industry.

**Consistent terminology.** To stay consistent, we use the terms message and frame interchangeably because we only consider messages that fit into one frame (maximum 8 bytes). For the purpose of using simple notation, we call a CAN message as periodic, sporadic or mixed if it is queued by an application task that is invoked periodically, sporadically or both (periodically and

sporadically) respectively. If a message is queued for transmission at periodic intervals, we use the term "Period" to refer to its periodicity. A sporadic message is queued for transmission as soon as an event occurs that changes the value of one or more signals contained in the message provided the Minimum Update Time ($MUT$) between queueing of two successive sporadic messages has elapsed. Hence, the transmission of a sporadic frame is constrained by the $MUT$. We overload the term "$MUT$" to refer to the "Inhibit Time" in CANopen protocol [41] and the "Minimum Delay Time (MDT)" in AUTOSAR communication [42].

### 5.3.1 Method 1: Implementation in the CANopen Protocol

The CANopen protocol [41] supports mixed transmission that corresponds to the Asynchronous Transmission Mode coupled with the Event Timer. The Event Timer is used to transmit an asynchronous message cyclically. A mixed message can be queued for transmission at the arrival of an event provided the 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 the Event Timer. The Event Timer is reset every time the message is queued. Once a mixed message is queued, any additional queueing of it will not take place during the Inhibit Time [41].

The transmission pattern of a mixed message in CANopen is illustrated in Figure 5.2. The down-pointing arrows symbolize the queueing of messages while the upward lines (labeled with alphabetic characters) represent arrival of the events. Message 1 is queued as soon as the event $A$ arrives. Both the Event Timer and Inhibit Time are reset. As soon as the Event Timer expires, message 2 is queued due to periodicity and both the Event Timer and Inhibit Time are reset again. When the event $B$ arrives, message 3 is immediately queued because the Inhibit Time has already expired. Note that the Event Timer is also reset at the same time when message 3 is queued as shown in Figure 5.2. Message 4 is queued because of the expiry of the Event Timer. There exists a dependency relationship between the Inhibit Time and the Event Timer, i.e., the Event Timer is reset not only with every periodic transmission but also with every sporadic transmission.

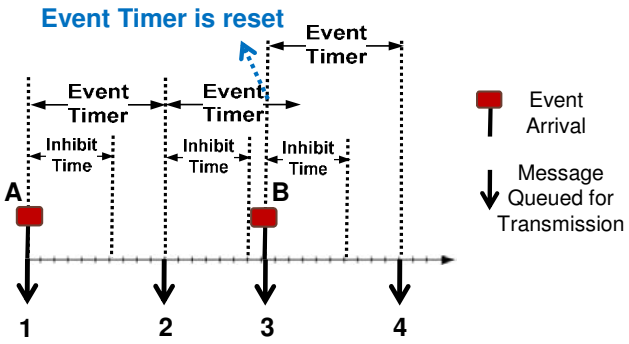


Figure 5.2: Mixed transmission pattern in the CANopen protocol

### 5.3.2 Method 2: Implementation in the AUTOSAR Communications

AUTOSAR (AUTomotive Open System ARchitecture) [43] can be viewed as a higher-level protocol if it uses CAN for network communication. Mixed transmission mode in AUTOSAR is widely used in practice. In AUTOSAR, a mixed message can be queued for transmission repeatedly with a period equal to the mixed transmission mode time period. The mixed message can also be queued at the arrival of an event provided the $MDT$ timer has been expired. However, each transmission of the mixed message, regardless of being periodic or sporadic, is limited by the $MDT$ timer. This means that both periodic and sporadic transmissions are delayed until the $MDT$ timer expires. The transmission pattern of a mixed message implemented by AUTOSAR is illustrated in Figure 5.3. Message $1$ is queued (the $MDT$ timer is started) because of partly periodic nature of the mixed message. When the event $A$ arrives, message $2$ is queued immediately because the $MDT$ timer has already expired. The next periodic transmission is scheduled $2$ time units after the transmission of message $2$. However, the next two periodic transmissions corresponding to messages $3$ and $4$ are delayed because the $MDT$ timer is not expired. This is indicated by the comment "Delayed Periodic Transmissions" in Figure 5.3. The periodic transmissions corresponding to messages $5$ and $6$ occur at the scheduled times because the $MDT$ timer is already expired in both cases.

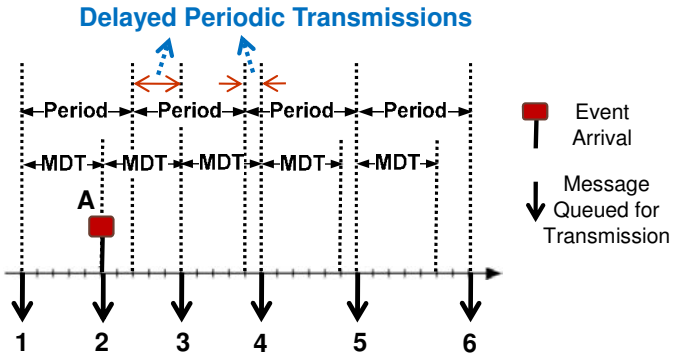


Figure 5.3: Mixed transmission pattern in the AUTOSAR Communications

### 5.3.3 Method 3: Implementation in the HCAN Protocol

A mixed message in the HCAN protocol [44] contains signals out of which some are periodic and some are sporadic. 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 the $MUT$ timer between the queueing of two successive sporadic instances of the mixed message has elapsed. Hence, the transmission of the mixed message due to arrival of events is constrained by the $MUT$ timer. The transmission pattern of the mixed message is illustrated in Figure 5.4. Message $1$ is queued because of periodicity. As soon as event $A$ arrives, message $2$ is queued. When event $B$ arrives it is not queued immediately because the $MUT$ timer is not expired yet. As soon as the $MUT$ timer expires, message $3$ is queued. Message $3$ contains the signal changes that correspond to event $B$. Similarly, a message is not immediately queued when the event $C$ arrives because the $MUT$ timer is not expired. Message $4$ is queued because of the periodicity. Although, the $MUT$ timer was not 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 sporadic message when the $MUT$ timer expires. This indicates that the periodic transmission of the mixed message cannot be interfered by its sporadic transmission (a unique property of the HCAN protocol). When the event $D$ arrives, a sporadic instance of the mixed message is immediately queued as message $5$ because the $MUT$ timer has already expired. Message $6$ is queued due to periodicity.

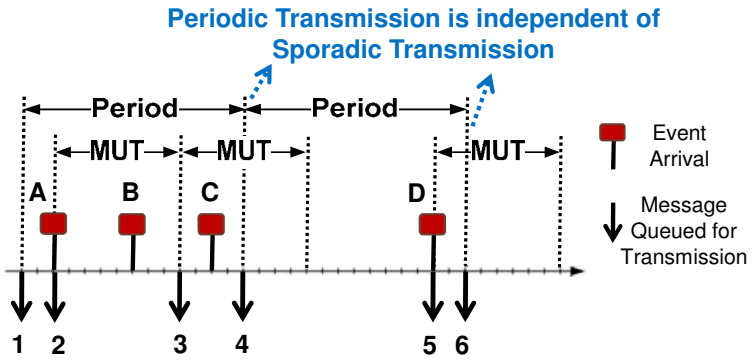


Figure 5.4: Mixed transmission pattern in the HCAN protocol

### 5.3.4 Discussion

In the first method [41], the Event Timer is reset every time the mixed message is queued for transmission. The implementation of mixed message in method 2 [42] is similar to method 1 to some extent. The main difference is that in method 2, the periodic transmission can be delayed until the expiry of the $MDT$ timer. Whereas in method 1, the periodic transmission is not delayed, in fact, the Event Timer is restarted with every sporadic transmission. The $MDT$ timer is started with every periodic or sporadic transmission of the mixed message. Hence, the worst-case periodicity of the mixed message in methods 1 and 2 can never be higher than the Inhibit Timer and $MDT$ timer respectively. This means that the models of mixed messages in the first and second implementation methods reduce to the classical sporadic model. Therefore, the existing analyses for CAN [16, 18, 21, 28] can be used for analyzing mixed messages in the first and second implementation methods.

However, the periodic transmission is independent of the sporadic transmission in the third method [44]. The periodic timer is not reset with every sporadic transmission. The mixed message can be queued for transmission even if the $MUT$ timer is not expired. Hence, the worst-case periodicity of the mixed message is neither bounded by period nor by the $MUT$ timer. Therefore, the analyses in [16, 18, 21, 28] cannot be used for analyzing the mixed messages in the third implementation method. This implies the need for the extension of existing analyses to support the mixed messages.

## 5.4 System Model

The system model is an extension of the model developed by Tindell et al. [16] and later extended in the works [28, 21]. The previous models do not support mixed messages. This model is extended in such a way that it supports mixed messages along with periodic and sporadic messages. The system consists of a number of CAN controllers (nodes[1]) denoted by $CC_1, CC_2, ...CC_n$. The nodes are connected to a single CAN network. The nodes implement priority-ordered queues, i.e., the highest priority message in each node enters into the bus arbitration. The set of messages in the system is defined as $\aleph$.

Let $\aleph_c$ defines the set of messages sent by the CAN controller $CC_c$. We assume that each CAN controller has a finite number of transmit buffers (however, each CAN controller is assumed to contain at least three transmit buffers). Let $K_c$ denote the transmit buffers in the CAN controller $CC_c$. The number of transmit buffers in $CC_c$ is returned by the function $Sizeof(K_c)$.

Each CAN message $m_m$ has a unique identifier and a priority denoted by $ID_m$ and $P_m$ respectively. The priority of a message is assumed to be equal to its ID. The priority of $m_m$ is considered higher than the priority of $m_n$ if $P_m < P_n$. Let the sets $hp(m_m)$, $lp(m_m)$, and $hep(m_m)$ contain the messages with priorities higher, lower, and equal and higher than $m_m$ respectively. Although the priorities of CAN messages are unique, the set $hep(m_m)$ will be used in the case of mixed messages.

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 the standard CAN frame uses an 11-bit identifier whereas the extended CAN frame uses a 29-bit identifier. In order to keep the notations simple and consistent, we define a function $\xi_m$ that denotes the transmission type of a message. $\xi_m$ specifies whether $m_m$ is periodic ($P$), sporadic ($S$) or mixed ($M$). Formally, the domain of $\xi_m$ can be defined as follows.

$$\xi_m \in [P, \quad S, \quad M]$$

Each message $m_m$ has a transmission time $C_m$ and queueing jitter $J_m$ which is inherited from the task that queues $m_m$, i.e., the sending task. We assume that $J_m$ can be smaller, equal or greater than $T_m$ or $MUT_m$. Each message can carry a data payload that ranges from 0 to 8 bytes. This integer value is specified in the header field of the frame called Data Length Code and

[1] we overload the terms node and CAN controller throughout the paper

is denoted by $s_m$. In the case of periodic transmission, $m_m$ has a transmission period which is denoted by $T_m$. Whereas in the case of sporadic transmission, $m_m$ has the $MUT_m$ time. $B_m$ denotes the blocking time of $m_m$ which refers to the largest amount of time $m_m$ has to wait for the transmission of a lower priority message.

We duplicate a message when its transmission type is mixed. Hence, each mixed message $m_m$ is treated as two separate messages, i.e., one periodic and the other sporadic. The duplicates share all the attributes except the $T_m$ and $MUT_m$. The periodic copy inherits $T_m$ while the sporadic copy inherits the $MUT_m$. 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). $m_m$ is deemed schedulable if its $R_m$ is less than or equal to its deadline $D_m$. The system is considered schedulable if all of its messages are schedulable.

We consider the deadlines to be arbitrary which means that they can be greater than the periods or $MUT$s of corresponding messages. We assume that the CAN controllers are capable of buffering more than one instance of a message. The instances of a message are assumed to be transmitted in the same order in which they are queued (i.e., we assume FIFO policy among the instances of the same message). For better readability, all the notations used in this paper are tabulated in Appendix A.

## 5.5 Extended Worst-case RTA for CAN without Buffer Limitations

In this section, we extend the existing RTA for CAN [16, 18] to support all types of messages namely periodic, sporadic, and mixed. However, we do not consider the buffer limitations in this section. That is, the CAN controllers are assumed to implement very large but finite number of transmit buffers such that there is no need to abort transmission requests[2]. The effect of buffer limitations will be integrated with the extended analysis in the subsequent sections.

Let $m_m$ be the message under analysis. First, we discuss few terms that are used in the extended analysis. In order to calculate the worst-case response time of $m_m$, the maximum busy period [16, 18] for priority level-m should be known.

[2]We use the term "no buffer limitations" consistently throughout the paper

**Maximum busy period.** It is the longest contiguous interval of time during which $m_m$ is unable to complete its transmission due to two reasons. First, the bus is occupied by the higher priority messages. In other words, at least one message of priority level-m or higher has not completed its transmission. Second, a lower priority message already started its transmission when $m_m$ is queued for transmission. The maximum busy period starts at the so-called critical instant.

**Critical instant.** For a system where messages are scheduled without offsets, the critical instant corresponds to the point in time when all higher priority messages in the system are assumed to be queued simultaneously with $m_m$ while their subsequent instances are assumed to be queued after the shortest possible interval of time [18].

We analyze $m_m$ differently based on its transmission type. Intuitively, we consider two different cases: (1) periodic and sporadic, and (2) mixed.

### 5.5.1 Case 1: When the Message Under Analysis ($m_m$) is Periodic or Sporadic

Consider $m_m$ to be a periodic or sporadic message. Since we consider arbitrary deadlines for messages, there can be more than one instance of $m_m$ that may become ready for transmission before the end of priority level-m busy period. There can be another reason to check if more than one instance of $m_m$ is queued for transmission in the priority level-m busy period. Since, the message transmission in CAN is non-preemptive, the transmission of previous instance of $m_m$ could delay the current instance of a higher priority message that may add to the interference received by the current instance of $m_m$. This phenomenon was identified by Davis et al. [18] and termed as "push-through interference". Because of this interference, a higher priority message may be waiting for its transmission before the transmission of the current instance of $m_m$ finishes. Hence, the length of busy period may extend beyond $T_m$ or $MUT_m$.

Therefore, we need to calculate the response time of each instance of $m_m$ within priority level-m busy period. The maximum value among the response times of all instances of $m_m$ is considered as the worst-case response-time of $m_m$. Let $q_m$ be the index variable to denote instances of $m_m$. The worst-case response time of $m_m$ is given by:

$$R_m = max\{R_m(q_m)\} \tag{5.1}$$

According to the existing analysis [16, 18], the worst-case response-time of any instance of $m_m$ consists of three parts as follows.

1. The queueing jitter denoted by $J_m$. It is inherited from the sending task, i.e., the task that queues $m_m$ for transmission. Basically, it represents the maximum variation in time between the release of the sending task and queuing of the message in the transmit queue (buffers). It is calculated by taking the difference between the worst- and best-case response times of the sending task.

2. The queueing delay denoted by $\omega_m$. It is equal to the longest time that elapses between the instant $m_m$ is queued by the sending task in the transmit queue and the instant when $m_m$ is about to start its successful transmission. In other words, $\omega_m$ is the interference caused by other messages to $m_m$.

3. The worst-case transmission time denoted by $C_m$. It represents the longest time it takes for $m_m$ to be transmitted over the network.

Thus, the worst-case response time of any instance $q_m$ of a periodic or sporadic message $m_m$ is given by the following set of equations.

$$R_m(q_m) = \begin{cases} J_m + \omega_m(q_m) - q_m T_m + C_m, & \text{if } \xi_k = \text{P} \\ J_m + \omega_m(q_m) - q_m MUT_m + C_m, & \text{if } \xi_k = \text{S} \end{cases} \tag{5.2}$$

The terms $q_m\, T_m$ and $q_m MUT_m$ in (5.2) are used to support the response-time calculations for multiple instances of $m_m$.

### Calculations for the Worst-case Transmission Time $C_m$

The worst-case transmission time of $m_m$ is calculated according to the method derived in [16] and later adapted by [18]. For the standard CAN identifier frame format, $C_m$ is calculated as follows.

$$C_m = \left(47 + 8s_m + \left\lfloor \frac{34 + 8s_m - 1}{4} \right\rfloor\right)\tau_{bit} \tag{5.3}$$

Where $\tau_{bit}$ denotes the time required to transmit a single bit of data on the CAN network. Its value depends upon the speed of the network. In (5.3), $47$ is the number of bits due to protocol overhead. It is composed of start of frame

bit (1-bit), arbitration field (12-bits), control field (6-bits), Cyclic Redundancy Check (CRC) field (16-bits), acknowledgement (ACK) field (2-bits), End of Frame (EoF) field (7-bits), and inter-frame space (3-bits). The number of bits due to protocol overhead in the case of extended CAN frame format is equal to $67$.

In [45], Broster identified that the analysis in [16, 18] uses $47$-bits instead of $44$-bits as the protocol overhead for a standard CAN identifier frame format. This is because the analysis in [16, 18] accounts 3-bit inter-frame space as part of the CAN frame. The 3-bit inter-frame space must be considered when calculating the interferences or blocking from other messages. However, Broster argued that this adds slight amount of pessimism to the response time of the message under analysis if the 3-bit inter-frame space is also considered in its transmission time. This is because the destination node can access the message before the inter-frame space. In order to avoid this pessimism, we subtract 3-bit time from the response time of the instance of the message under analysis.

The term $\left\lfloor \frac{34+8s_m-1}{4} \right\rfloor$ in (5.3) is added to compensate for the extra time due to *bit stuffing*. It should be noted that the bit sequences $000000$ and $111111$ are used for error signals in CAN. In order to be unambiguous in non-erroneous transmission, a stuff bit of opposite polarity is added whenever there are five bits of the same polarity in the sequence of bits to be transmitted [18]. The value $34$ indicates that only 34-bits out of 47-bits protocol overhead are subjected to bit stuffing. The term $\lfloor \frac{a}{b} \rfloor$ is the notation for *floor* function. It returns the largest integer that is less than or equal to $\frac{a}{b}$.

Similarly, $C_m$ is calculated for the extended CAN identifier frame format as follows.

$$C_m = \left(67 + 8s_m + \left\lfloor \frac{54 + 8s_m - 1}{4} \right\rfloor\right)\tau_{bit} \tag{5.4}$$

The calculations for $C_m$ in (5.3) can be simplified as follows.

$$C_m = (55 + 10s_m)\tau_{bit} \tag{5.5}$$

Similarly, the calculations for $C_m$ in (5.4) can be simplified as follows.

$$C_m = (80 + 10s_m)\tau_{bit} \tag{5.6}$$

**Calculations for the Queueing Delay $\omega_m$**

In (5.2), the queueing delay for any instance of $m_m$ consists of two elements.

**1) Blocking delay.** If any lower priority message just starts its transmission when $m_m$ is queued for transmission then $m_m$ has to wait in the transmit queue and is said to be blocked by the lower priority message. The lower priority message cannot be preempted during its transmission because CAN uses fixed-priority non-preemptive scheduling. Since we consider arbitrary deadline, $m_m$ can also be blocked from its own previous instance due to push-through blocking [18] as discussed in Subsection 5.5.1. It should be noted that a CAN message can be blocked either by only one message in the set of lower priority messages or by only one of its previous instances. Moreover, the message under analysis can only be blocked by either the periodic copy or the sporadic copy of any lower priority mixed message (both copies of a mixed message have the same transmission time, $C_m$). Therefore, the maximum blocking delay is equal to the largest transmission time in the set of lower priority messages including the message itself. The maximum blocking delay for $m_m$ denoted by $B_m$ is calculated as follows.

$$B_m = \max_{\forall m_j \in lep(m_m)} \{C_j\} \tag{5.7}$$

Since we consider arbitrary deadlines, $m_m$ can also be blocked from its own previous instance due to push-through blocking [18] as discussed in Subsection 5.5.1. That is the reason why (5.7) includes the function $lep(m_m)$ instead of $lp(m_m)$. It is important to point out that the blocking delay for the lowest priority message in the system is equal to zero if (5.7) is used. However, Broster [45] identified that lowest priority message can be blocked for 3-bits of time due to inter-frame space before it. Therefore, we consider $3\tau_{bit}$ time as the blocking delay for only the lowest priority message.

**2) Delay due to interference from higher priority messages.** Since CAN uses fixed-priority non-preemptive scheduling, a message cannot be interfered by higher priority messages during its transmission. Whenever we use the term interference, it refers to the amount of time $m_m$ has to wait in the transmit queue because the higher priority messages in the system win the arbitration, i.e., the right to transmit before $m_m$. We adapt the calculations for the interference from higher priority messages from the existing analysis [16, 18]. However, the existing analysis considers the interference from only higher-priority periodic or sporadic messages. As we discussed in the system model that a mixed message is duplicated as two messages (one periodic and the other sporadic), each higher-priority mixed message should contribute interference from both the duplicates.

Thus, the queueing delay sums up the interferences due to higher priority messages, previous instances of the same message and the blocking delay. The queueing delay $\omega_m$ for the instance $q_m$ of $m_m$ can be calculated by solving the following equation.

$$\omega_m^{n+1}(q) = B_m + q_m C_m + \sum_{\forall m_k \in hp(m_m)} I_k C_k \tag{5.8}$$

(5.8) is an iterative equation. It is solved iteratively until two consecutive solutions become equal or the solution exceeds the message deadline in which case the message is deemed unschedulable. The starting value for $\omega_m^n$ can be selected equal to $B_m + q_m C_m$. In (5.8), $I_k$ is calculated differently for different values of $\xi_k$ (k is the index of any higher priority message) as shown below.

$$I_k = \begin{cases} \left\lceil \frac{\omega_m^n(q_m)+J_k+\tau_{bit}}{T_k} \right\rceil, & \text{if } \xi_k = \text{P} \\ \left\lceil \frac{\omega_m^n(q_m)+J_k+\tau_{bit}}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{S} \\ \left\lceil \frac{\omega_m^n(q_m)+J_k+\tau_{bit}}{T_k} \right\rceil + \left\lceil \frac{\omega_m^n(q_m)+J_k+\tau_{bit}}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{M} \end{cases} \tag{5.9}$$

The term $\lceil \frac{a}{b} \rceil$ is the notation for *ceil* function. It returns the smallest integer that is greater than or equal to $\frac{a}{b}$. The three terms $\left\lceil \frac{\omega_m^n(q_m)+J_k+\tau_{bit}}{T_k} \right\rceil$, $\left\lceil \frac{\omega_m^n(q_m)+J_k+\tau_{bit}}{MUT_k} \right\rceil$ and $\left[ \left\lceil \frac{\omega_m^n(q_m)+J_k+\tau_{bit}}{T_k} \right\rceil + \left\lceil \frac{\omega_m^n(q_m)+J_k+\tau_{bit}}{MUT_k} \right\rceil \right]$ in (5.9) represent the maximum number of instances of higher-priority periodic, sporadic and mixed messages that are queued for transmission in the maximum busy period respectively. It is evident that the interference from a higher-priority mixed message contains the contribution from both of its duplicates.

### Calculations for the Length of Priority Level-m Busy Period

In order to calculate the worst-case response time of $m_m$, the number of instances of $q_m$ that become ready for transmission before the end of the priority level-m busy period should be known. The length of the priority level-m busy

period, denoted by $t_m$, can be calculated by adapting the existing analysis [18] as follows.

$$t_m^{n+1} = B_m + \sum_{\forall m_k \in hep(m_m)} I'_k C_k \tag{5.10}$$

where $I'_k$ is given by the following relation. Note that the contribution from both the duplicates of the mixed message $m_k$ is taken into account, provided it belongs to the set of equal or higher priority messages with respect to $m_m$.

$$I'_k = \begin{cases} \left\lceil \frac{t_m^n + J_k}{T_k} \right\rceil, & \text{if } \xi_k = \text{P} \\ \left\lceil \frac{t_m^n + J_k}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{S} \\ \left\lceil \frac{t_m^n + J_k}{T_k} \right\rceil + \left\lceil \frac{t_m^n + J_k}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{M} \end{cases} \tag{5.11}$$

In order to solve the iterative equation (5.10), $C_m$ can be used as the initial value of $t_m^n$. The right hand side of (5.10) is a monotonic non-decreasing function of $t_m$. The iterative equation (5.10) 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,

$$U_m < 1 \tag{5.12}$$

where $U_m$ is calculated as follows.

$$U_m = \sum_{\forall m_k \in hep(m_m)} C_k I''_k \tag{5.13}$$

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

$$I''_k = \begin{cases} \frac{1}{T_k}, & \text{if } \xi_k = \text{P} \\ \frac{1}{MUT_k}, & \text{if } \xi_k = \text{S} \\ \frac{1}{T_k} + \frac{1}{MUT_k}, & \text{if } \xi_k = \text{M} \end{cases} \tag{5.14}$$

In the above equation, the contribution from both copies of all mixed messages that are included in the set of equal and higher priority messages with respect to $m_m$ is taken into account while calculating the bus utilization.

**Calculations for the number of instances of $m_m$.** The number of instances of $m_m$, denoted by $Q_m$, that become ready for transmission before the busy period ends is calculated as follows.

$$Q_m = \begin{cases} \left\lceil \frac{t_m + J_m}{T_m} \right\rceil, & \text{if } \xi_k = \text{P} \\ \left\lceil \frac{t_m + J_m}{MUT_m} \right\rceil, & \text{if } \xi_k = \text{S} \end{cases} \tag{5.15}$$

The range for the index variable $q_m$ for the number of instances of $m_m$ queued in the priority level-m busy period is given as follows.

$$0 \leq q_m \leq Q_m - 1 \tag{5.16}$$

The response times of all instances of $m_m$ in the range shown in (5.16) should be calculated while the largest among them represents the worst-case response time of $m_m$.

### 5.5.2 Case 2: When the Message Under Analysis is Mixed

When the message under analysis is mixed, we treat the message as two separate message streams, i.e., the mixed message is duplicated as the periodic and sporadic messages. The response time of both the duplicates is calculated separately. Consider $m_m$ to be a mixed message. For simplicity, we denote the periodic and sporadic copies of $m_m$ by $m_{m_P}$ and $m_{m_S}$ respectively. Let the worst-case response times of $m_{m_P}$ and $m_{m_S}$ are denoted by $R_{m_P}$ and $R_{m_S}$ respectively. The worst-case response time of $m_m$ is the largest value between $R_{m_P}$ and $R_{m_S}$ as given by the following equation.

$$R_m = max\{R_{m_P}, \ \ R_{m_S}\} \tag{5.17}$$

Where $R_{m_P}$ and $R_{m_S}$ are equal to the maximum value among the response times of their respective instances. Let $q_{m_P}$ and $q_{m_S}$ be the index variables to denote the instances of $m_{m_P}$ and $m_{m_S}$ respectively. The calculations for $R_{m_P}$ and $R_{m_S}$ are adapted from the periodic and sporadic cases (discussed in the previous subsection) respectively as follows.

$$R_{m_P} = max\{R_{m_P}(q_{m_P})\}, \qquad \forall\, 0 \leq q_{m_P} \leq (Q_{m_P} - 1) \tag{5.18}$$

$$R_{m_S} = max\{R_{m_S}(q_{m_S})\}, \qquad \forall\, 0 \leq q_{m_S} \leq (Q_{m_S} - 1) \tag{5.19}$$

Where, $Q_{m_P}$ and $Q_{m_S}$ represent the number of instances of $m_{m_P}$ and $m_{m_S}$ that are queued in the priority level-m busy period respectively. We will come back to these two terms later in this subsection.

In (5.18) and (5.19), $R_{m_P}$ and $R_{m_S}$ for each respective instance are calculated separately by adapting the response-time calculations for the periodic and sporadic messages (from previous subsection) as follows.

$$R_{m_P}(q_{m_P}) = J_m + \omega_{m_P}(q_{m_P}) - q_{m_P}T_m + C_m \tag{5.20}$$

$$R_{m_S}(q_{m_S}) = J_m + \omega_{m_S}(q_{m_S}) - q_{m_S}MUT_m + C_m \tag{5.21}$$

The queueing jitter, $J_m$, is the same (equal) in both the equations (5.20) and (5.21). The worst-case transmission time, $C_m$, is also the same in these equations and is calculated using (5.5) or (5.6) depending upon the type of CAN frame identifier. Although, $m_{m_P}$ and $m_{m_S}$ inherit same $J_m$ and $C_m$ from $m_m$, they experience different amount of queueing delay caused by other messages.

### Calculations for the Queueing Delay

The queueing delay experienced by $m_{m_P}$ and $m_{m_S}$ is denoted by $\omega_{m_P}$ and $\omega_{m_S}$ in (5.20) and (5.21) respectively. $\omega_{m_P}$ and $\omega_{m_S}$ can be calculated by adapting the calculations for the queueing delay in (5.8). However, in this equation we need to add the effect of self interference in a mixed message. By self interference, we mean that the periodic copy of a mixed message can be interfered by the sporadic copy and vice versa. Since, both $m_{m_P}$ and $m_{m_S}$ have equal priorities, any instance of $m_{m_S}$ queued ahead of $m_{m_P}$ will contribute an extra delay to the queueing delay experienced by $m_{m_P}$. A similar argument holds in the case of $m_{m_S}$. Let the self interference experienced by $m_{m_P}$ due to one or more instances of $m_{m_S}$ be denoted by $SI^P_{m_S}$. Similarly, $SI^S_{m_P}$ represents the self interference experienced by $m_{m_S}$ due to one or more instances of $m_{m_P}$. Hence, $\omega_{m_P}$ and $\omega_{m_S}$ can be calculated as follows.

$$\omega^{n+1}_{m_P}(q_{m_P}) = B_m + q_{m_P}C_m + \sum_{\forall m_k \in hp(m_m)} I_{k_P}C_k + SI^P_{m_S} \tag{5.22}$$

$$\omega_{m_S}^{n+1}(q_{m_S}) = B_m + q_{m_S}C_m + \sum_{\forall m_k \in hp(m_m)} I_{k_S}C_k + SI_{m_P}^S \qquad (5.23)$$

**Calculations for the self interference in a mixed message.** In order to derive the contribution of one copy of a mixed message to the queueing delay of the other, consider three different cases, depicting the transmission pattern of a mixed message $m_m$, shown in Figure 5.5. In the first case, we assume $T_m$ to be greater than $MUT_m$. That is, there can be more transmissions of $m_{m_S}$ compared to that of $m_{m_P}$. Since, the maximum update time between the queueing of any two instances of $m_{m_S}$ can be arbitrarily very long, it is possible to have fewer sporadic transmissions than periodic transmissions of $m_m$. In the second case, we assume that $T_m$ is equal to $MUT_m$. In this case, there are equal number of transmissions of $m_{m_P}$ and $m_{m_S}$. In the third case, we assume $T_m$ to be smaller than $MUT_m$. This implies that the number of sporadic transmissions will be less than the periodic transmissions of $m_m$.

It is important to note that in the example shown in Figure 5.5, there is a small offset between the first periodic and sporadic transmission of $m_m$. This offset is used to maximize the queueing delay. If this offset is removed then only one message will be queued corresponding to the first instance of both $m_{m_P}$ and $m_{m_S}$. 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 sporadic transmissions of $m_m$ will not overlap, there by, maximizing the queueing delay.

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

Let the message under analysis be $m_{m_P}$ and consider case (a) in Figure 5.5. An application task queues $m_m$ periodically with a period $T_m$ (equal to 9 time units). Moreover, the same task can also queue $m_m$ sporadically at the arrival of events (labeled with numbers 1-6). The queueing of $m_{m_S}$ is constrained by $MUT_m$ (equal to 3 time units). The first instance of $m_{m_P}$ ($q_{m_P} = 0$) is queued for transmission as shown by $m_{m_P}(0)$ in Figure 5.5. If event *1* had arrived at the same time as the queueing of $m_{m_P}(0)$ then the signals in $m_{m_S}(0)$ would have been updated as part of $m_{m_P}(0)$. In that case, $m_{m_S}(0)$ would not have been queued separately (this is the property of the mixed message in the HCAN protocol). In order to maximize the contribution of $m_{m_S}$ on the queueing delay of $m_{m_P}$, $m_{m_S}(0)$ is queued just after the queueing of $m_{m_P}(0)$ as shown in all the cases in Figure 5.5. Therefore, $m_{m_S}(0)$ and subsequent instances of $m_{m_S}$

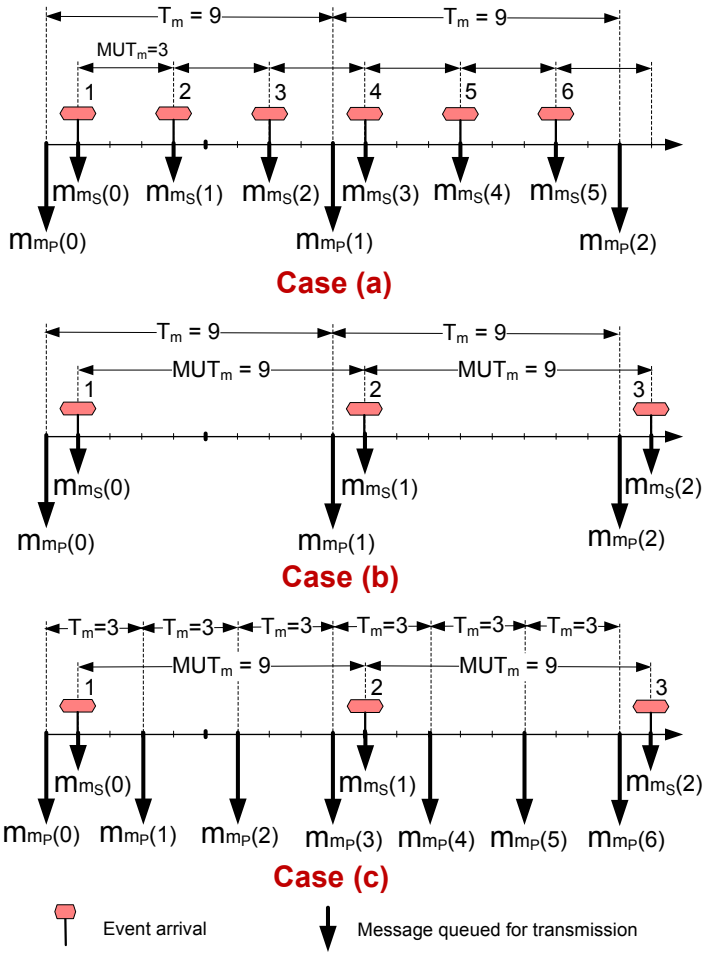


Figure 5.5: Self interference in a mixed message: (a) $\mathbf{T_m > MUT_m}$, (b) $\mathbf{T_m = MUT_m}$, (c) $\mathbf{T_m < MUT_m}$

.

will have no contribution in the worst-case queueing delay of the first instance of $m_{m_P}$ denoted by $m_{m_P}(0)$.

Consider the second instance of $m_{m_P}$. All those instances of $m_{m_S}$ that

are queued ahead of $m_{m_P}(1)$ will contribute to its queueing delay. It can be observed in the case (a) that the first three instances of $m_{m_S}$ are queued ahead of $m_{m_P}(1)$. Similarly, there are six instances of $m_{m_S}$ that are queued ahead of $m_{m_P}(2)$.

Let $Q^P_{m_S}$ denotes the total number of instances of $m_{m_S}$ that are queued ahead of the $q^{th}_{m_P}$ instance of $m_{m_P}$. We can generalize $Q^P_{m_S}$ for the case (a) as follows.

$$Q^P_{m_S} = \left\lceil \frac{q_{m_P} T_m}{MUT_m} \right\rceil \tag{5.24}$$

For example, consider again the queueing of different instances of $m_{m_S}$ and $m_{m_P}$ in the case (a). Equation (5.24) yields the set $\{Q^P_{m_S} = 0, 3, 6, ...\}$ for the corresponding values in the set $\{q_{m_P} = 0, 1, 2, ...\}$. Thus the total number of instances of $m_{m_S}$ queued ahead of each instance of $m_{m_p}$ calculated by (5.24) are consistent with the case (a) in Figure 5.5.

**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 5.5 that there are 0, 1, and 2 instances of $m_{m_S}$ that are queued ahead of $m_{m_P}(0)$, $m_{m_P}(1)$ and $m_{m_P}(2)$ respectively. When (5.24) is applied in case (b), we get the set $\{Q^P_{m_S} = 0, 1, 2, ...\}$ for the corresponding values in the set $\{q_{m_P} = 0, 1, 2, ...\}$. Therefore, (5.24) is also applicable on case (b).

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

Now, consider case (c) in which $T_m$ (3 time units) is smaller than $MUT_m$ (9 time units). The first instance of $m_{m_S}$ denoted by $m_{m_S}(0)$ will be queued ahead of $m_{m_P}(1)$, $m_{m_P}(2)$ and $m_{m_P}(3)$. Similarly, the two instances of $m_{m_S}$ denoted by $m_{m_S}(0)$ and $m_{m_S}(1)$ will contribute to the queueing delay of $m_{m_P}(4)$, $m_{m_P}(5)$ and $m_{m_P}(6)$. (5.24) yields the set $\{Q^P_{m_S} = 0, 1, 1, 1, 2, 2, 2, 2, ...\}$ for the corresponding values in the set $\{q_{m_P} = 0, 1, 2, 3, 4, 5, 6, ...\}$. Thus the total number of instances of $m_{m_S}$ queued ahead of each instance of $m_{m_P}$ calculated by (5.24) are consistent with the case (c) in Figure 5.5.

Now we consider the effect of jitter on the instances of $m_{m_S}$ prior to $m_{m_S}(0)$ which can be queued just ahead of $m_{m_P}(0)$ and contribute to the queueing delay of $m_{m_P}$. We assume FIFO queueing policy among the instances of the same message. By considering the jitter of $m_{m_S}$ in $Q^P_{m_S}$, (5.24) can be generalized for the three cases as follows.

$$Q^P_{m_S} = \left\lceil \frac{q_{m_P} T_m + J_m}{MUT_m} \right\rceil \tag{5.25}$$

The self interference experienced by $m_{m_P}$ due to one or more instances of $m_{m_S}$ is the product of $Q^P_{m_S}$ and worst-case transmission time of $m_{m_S}$ as follows.

$$SI^P_{m_S} = Q^P_{m_S} C_m = \left\lceil \frac{q_{m_P} T_m + J_m}{MUT_m} \right\rceil C_m \tag{5.26}$$

The total number of instances of $m_{m_P}$ that are queued ahead of the $q^{th}_{m_S}$ instance of $m_{m_S}$, denoted by $Q^S_{m_P}$, can be derived in a similar fashion. Thus, $Q^S_{m_P}$ can be calculated by the following equation.

$$Q^S_{m_P} = \left\lceil \frac{q_{m_S} MUT_m + J_m}{T_m} \right\rceil \tag{5.27}$$

Once again, the self interference experienced by $m_{m_S}$ due to one or more instances of $m_{m_P}$ is the product of $Q^S_{m_P}$ and worst-case transmission time of $m_{m_P}$ as follows.

$$SI^S_{m_P} = Q^S_{m_P} C_m = \left\lceil \frac{q_{m_S} MUT_m + J_m}{T_m} \right\rceil C_m \tag{5.28}$$

From (5.26) and (5.28) it is obvious that when $q_{m_P}$ and $q_{m_S}$ are zero (i.e., zeroth instances of $m_{m_P}$ and $m_{m_S}$) as well as $J_m$ is also zero then $SI^P_{m_S}$ and $SI^S_{m_P}$ are also zero respectively. However, even if $J_m$ is zero, the zeroth instance of $m_{m_P}$ can be interfered by one instance of $m_{m_S}$. Similar argument holds for the zeroth instance of $m_{m_E}$. For example, consider Case (a) in Figure 5.5. Let $m_m$ be the highest priority message. Let $m_{m_S}(0)$ is queued just after the queueing of $m_{m_P}(0)$. The instance $m_{m_P}(0)$ can be blocked by any lower priority message. However, $m_{m_S}(0)$ cannot start its transmission unless $m_{m_P}(0)$ is transmitted. Therefore, we have to consider this specific case for the calculation of self interference in (5.26) and (5.28) as follows. This specific case is not considered for the calculations of self interference in [1]. It should be noted that this specific case may not occur if we consider holistic view of a distributed system using CAN network. This is because a message inherits its release jitter (most often non-zero) that is equal to the difference between worst- and best-case response times of the sending task.

$$SI^P_{m_S} = \begin{cases} \left\lceil \frac{q_{m_P} T_m + J_m + \tau_{bit}}{MUT_m} \right\rceil C_m, & \text{if } (q_{m_P} = 0) \,\&\&\, (J_m = 0) \\ \left\lceil \frac{q_{m_P} T_m + J_m}{MUT_m} \right\rceil C_m, & \text{otherwise} \end{cases} \tag{5.29}$$

$$SI^S_{m_P} = \begin{cases} \left\lceil \frac{q_{m_S} MUT_m + J_m + \tau_{bit}}{T_m} \right\rceil C_m, & \text{if } (q_{m_S} = 0) \;\&\&\; (J_m = 0) \\ \left\lceil \frac{q_{m_S} MUT_m + J_m}{T_m} \right\rceil C_m, & \text{otherwise} \end{cases} \tag{5.30}$$

(5.22) and (5.23) are solved iteratively until two consecutive solutions of each equation become equal or the solution exceeds the message deadline in which case the message is deemed unschedulable. The starting values for $\omega^n_{m_P}$ and $\omega^n_{m_S}$ can be selected equal to $B_m + q_{m_P} C_m$ and $B_m + q_{m_S} C_m$ respectively. The blocking time $B_m$ is calculated using (5.7). The calculations for $I_{k_P}$ and $I_{k_S}$ are adapted from (5.9) separately for $m_{m_P}$ and $m_{m_S}$ as follows.

$$I_{k_P} = \begin{cases} \left\lceil \frac{\omega^n_{m_P}(q_{m_P}) + J_k + \tau_{bit}}{T_k} \right\rceil, & \text{if } \xi_k = \text{P} \\ \left\lceil \frac{\omega^n_{m_P}(q_{m_P}) + J_k + \tau_{bit}}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{S} \\ \left\lceil \frac{\omega^n_{m_P}(q_{m_P}) + J_k + \tau_{bit}}{T_k} \right\rceil + \left\lceil \frac{\omega^n_{m_P}(q_{m_P}) + J_k + \tau_{bit}}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{M} \end{cases} \tag{5.31}$$

$$I_{k_S} = \begin{cases} \left\lceil \frac{\omega^n_{m_S}(q_{m_S}) + J_k + \tau_{bit}}{T_k} \right\rceil, & \text{if } \xi_k = \text{P} \\ \left\lceil \frac{\omega^n_{m_S}(q_{m_S}) + J_k + \tau_{bit}}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{S} \\ \left\lceil \frac{\omega^n_{m_S}(q_{m_S}) + J_k + \tau_{bit}}{T_k} \right\rceil + \left\lceil \frac{\omega^n_{m_S}(q_{m_S}) + J_k + \tau_{bit}}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{M} \end{cases} \tag{5.32}$$

### Calculations for the Length of Priority Level-m Busy Period

The length of priority level-m busy period, denoted by $t_m$, can be calculated using (5.10) that was developed for the periodic and sporadic messages. This is because (5.10) takes into account the effect of queueing delay from all 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 (5.10). Therefore, there is no need to calculate $t_m$ for $m_{m_P}$ and $m_{m_S}$ separately. In fact, $t_m$ should be calculated only once for the mixed message that is under analysis.

Although the length of priority level-m busy period is the same for $m_{m_P}$ and $m_{m_S}$, the number of instances of both these messages that become ready for transmission just before the end of the busy period, denoted by $Q_{m_P}$ and $Q_{m_S}$ respectively, may be different. The reason is that the calculations for $Q_{m_P}$ and $Q_{m_S}$ require $T_m$ and $MUT_m$ respectively and which may have different values. $Q_{m_P}$ and $Q_{m_S}$ can be calculated by adapting (5.15) that was derived for the calculations for the number of instances of periodic and sporadic messages. $Q_{m_P}$ and $Q_{m_S}$ are given by the following equations.

$$Q_{m_P} = \left\lceil \frac{t_m + J_m}{T_m} \right\rceil \tag{5.33}$$

$$Q_{m_S} = \left\lceil \frac{t_m + J_m}{MUT_m} \right\rceil \tag{5.34}$$

## 5.6 Integrating the Effect of Abortable Transmit Buffers with the Extended Worst-case RTA for CAN

In this section, we integrate the effect of abortable transmit buffers in the CAN controllers with the extended RTA of CAN for periodic, sporadic and mixed messages. We assume that the CAN controllers implement limited number of transmit buffers (at least three) and support transmission abort requests.

### 5.6.1 Priority Inversion in the Case of Abortable Transmit Buffers

If the CAN controller supports transmission abort requests (and implements more than 3 transmit buffers) then the lowest priority message in the transmit buffer (that is not undergoing transmission) is swapped with the higher priority message from the message queue. During the swapping process, a lower priority message from the transmit buffer in any other controller may win the bus arbitration and contribute an extra delay to the response time of the higher priority message. The copying delay and the extra blocking delay during the

swapping process should be taken into account while calculating the response time of the higher priority message.

**Additional delay and jitter due to priority inversion.** In order to demonstrate the additional delay due to priority inversion when CAN controllers support transmission abort requests, consider the example of transmission of a message set as shown in Figure 5.6. Assume there are three nodes $CC_c$, $CC_j$ and $CC_k$ in the system and each node has three transmit buffers. $m_1$ is the highest priority message in the node $CC_c$ as well as in the system.

When $m_1$ becomes ready for transmission in the message queue, a lower priority message $m_6$ belonging to node $CC_k$ is already under transmission. This represents the blocking delay for $m_1$. At this point in time, all transmit buffers in $CC_c$ are occupied by the lower priority messages (say $m_3$, $m_4$ and $m_5$). The device drivers signal an abort request for the lowest priority message in $K_c$ (transmit buffers in $CC_c$) that is not under transmission.

Hence, $m_5$ is aborted and copied from the transmit buffer to the message queue whereas $m_1$ is moved to the vacated transmit buffer. The time required to do this swapping is identified as *swapping time* in Figure 5.6. During the swapping time a series of events occur: $m_6$ finishes its transmission, new arbitration round starts, another message $m_2$ belonging to node $CC_j$ and having priority lower than $m_1$ wins the arbitration and starts its transmission. Thus $m_1$ has to wait in the transmit buffer until $m_2$ finishes its transmission. This results in the priority inversion and adds an extra delay to the response time of $m_1$. In [21], Khan et al. pointed out that this extra delay of the higher priority message appears as its additional jitter to the lower priority messages, e.g., $m_5$ in Figure 5.6.

**Calculations for the additional jitter.** The calculations for the additional jitter are adapted from the analysis in [21]. Let $m_m$ be the message under analysis that belongs to the node $CC_c$. Let $K_c$ denote the transmit buffer queue in $CC_c$. Let $CT_m$ denotes the maximum between the time required to copy $m_m$ from the message queue to the transmit buffer and from transmit buffer to the message queue. As noted in [21], these two times are very similar to each other in practice. Let the additional jitter of $m_m$ as seen by the lower priority messages due to priority inversion (discussed above) be denoted by $AJ_m^A$. Where $AJ$ stands for "Additional Jitter" while the superscript "A" stands for Abortable transmit buffer. The maximum jitter of $m_m$ denoted by $\hat{J}_m$ is the summation of its original jitter $J_m$ and the additional jitter due to priority inversion. Mathematically, the additional jitter of $m_m$ that is seen by lower priority messages is calculated as follows.

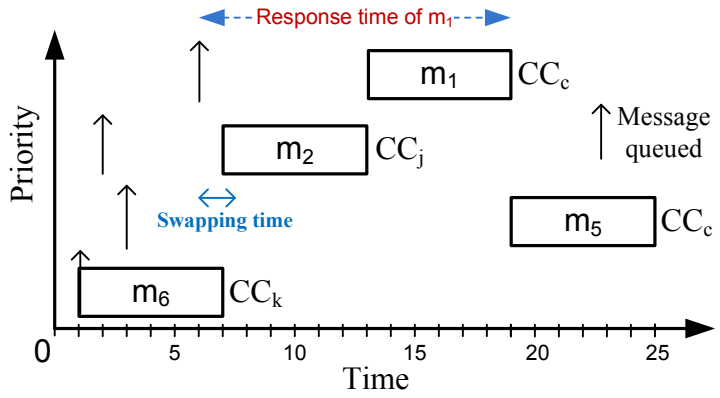


Figure 5.6: Demonstration of priority inversion in the case of abortable transmit buffers

$$\hat{J}_m = J_m + AJ_m^A \tag{5.35}$$

The additional jitter for $m_m$ depends upon the following three elements.

1. The largest copy time of a message in the set of lower priority messages that belong to the same node $CC_c$.

2. The largest value among the worst-case transmission times of all those messages whose priorities are lower than the priority of $m_m$ but higher than the highest priority message in $K_c$.

3. Since the original blocking time $B_m$ for $m_m$ is separately considered as part of the queueing delay, it should be subtracted from the additional delay.

Therefore, $AJ_m^A$ is calculated as follows:

$$AJ_m^A = max(0, \max_{\forall m_l \in CC_c \wedge m_l \in lep(m_m)}(CT_l) + \max_{P_m < P_l \leq P_{h_{K_c}}}(C_l) - B_m) \tag{5.36}$$

where $m_{h_{K_c}}$ is the highest priority message in $K_c$. We will come back to the calculations for finding the priority of $m_{h_{K_c}}$ in the next subsection.

**Calculations for the blocking delay.** When $m_m$ is subjected to priority inversion, it experiences an extra amount of blocking in addition to the original blocking delay $B_m$. Let the total blocking delay for $m_m$ due to priority inversion be denoted by $\hat{B}_m$. Mathematically, it is equal to the sum of the original blocking delay and the largest copy time of a message in the set of lower priority messages that belong to the same node $CC_c$.

$$\hat{B}_m = \max_{\forall m_j \in lep(m_m)} \{C_j\} + \max_{\forall m_l \in CC_c \wedge m_l \in lep(m_m)} (CT_l) \tag{5.37}$$

Since we consider arbitrary deadlines, $m_m$ can also be blocked from its own previous instance due to push-through blocking [18] as discussed in Subsection 5.5.1. That is the reason why (5.37) includes the function $lep(m_m)$ instead of $lp(m_m)$.

### 5.6.2 Extended RTA

The work in [21] noted that not all messages in a node suffer from priority inversion. Therefore we consider two different cases for calculating response times of periodic, sporadic and mixed messages in CAN with abortable transmit buffers. In this section, first we determine which messages are free from priority inversion. After that we extend the analysis from Section 5.5 by adapting the analysis in [21].

#### Calculations for the Number of Messages Free from Priority Inversion

If we assume that multiple instances of a message cannot occupy transmit buffers then the number of lowest priority messages equal to the number of transmit buffers in a node will be safe from priority inversion. Whereas, the rest of the messages in the same node may suffer from priority inversion. This can be explained by a simple but intuitive example. Let there be 4 transmit buffers in a node. Let there be 6 messages $m_1$, $m_2$, $m_3$, $m_4$, $m_5$ and $m_6$ in this node. $m_1$ has the highest priority, while $m_6$ has the lowest priority. Assume $m_3$ arrives in the message queue when 3 out of 4 transmit buffers are occupied by the three lowest priority messages $m_6$, $m_5$ and $m_4$. The fourth transmit buffer can either be empty or occupied by one of the higher priority messages $m_1$ or $m_2$. If the fourth transmit buffer is empty then $m_3$ is immediately copied to it. On the other hand, $m_3$ has to wait in the message queue because at least one transmit buffer contains a higher priority message. In both

cases there is no need to abort any transmission. This implies that $m_6$, $m_5$, $m_4$ and $m_3$ will be safe from priority inversion, whereas $m_1$ and $m_2$ may undergo priority inversion. In this case, $m_{h_{K_c}}$ is represented by message $m_3$.

This means that the set of lower priority messages whose size is equal to the number of transmit buffers will be free from priority inversion. However, this condition may become invalid if we assume that multiple instances of a message can occupy transmit buffers at the same time. In that case we need to find out the worst-case scenario where messages are free from priority inversion.

**Worst-case scenario for** $m_{h_{K_c}}$. For convenience, assume that $N_c$ represents the number of messages sent by the node $CC_c$. Intuitively, we can assume that the lowest priority message belonging to $CC_c$ can be indexed as $m_m^{N_c-1}$. It should be noted that $N_c - 1$ does not represent the priority of the message. Similarly, the second lowest priority message belonging to $CC_c$ can be indexed as $m_m^{N_c-2}$.

Let the total number of instances of all messages occupying the transmit buffers in $CC_c$ be denoted by $\Omega_c$. Assume that the maximum number of instances of $m_m^{N_c-1}$ occupying transmit buffers ahead of $m_m^{N_c-2}$ is denoted by $\Omega_c^{N_c-1_2}$. Its value depends upon three factors.

1. Periods of these two messages. If the period of $m_m^{N_c-2}$ is higher than the period of $m_m^{N_c-1}$, there can be more than one instance of $m_m^{N_c-1}$ that may occupy transmit buffers in $CC_c$ ahead of $m_m^{N_c-2}$.

2. Due to jitter of $m_m^{N_c-1}$, more than one instance of $m_m^{N_c-1}$ may occupy transmit buffers in $CC_c$.

3. Transmission type of $m_m^{N_c-1}$. If $m_m^{N_c-1}$ is a mixed message, we need to consider the contribution of its periodic as well as sporadic part.

The value of $\Omega_c^{N_c-1_2}$ can be calculated with a similar intuition that we used in (5.25) as follows.

$$\Omega_c^{N_c-1_2} = \begin{cases} \left\lceil \frac{{T_m}^{N_c-2} + {J_m}^{N_c-1}}{{T_m}^{N_c-1}} \right\rceil, & \text{if } {\xi_m}^{N_c-1} = \text{P} \\ \left\lceil \frac{{T_m}^{N_c-2} + {J_m}^{N_c-1}}{MUT_m{}^{N_c-1}} \right\rceil, & \text{if } {\xi_m}^{N_c-1} = \text{S} \\ \left\lceil \frac{{T_m}^{N_c-2} + {J_m}^{N_c-1}}{{T_m}^{N_c-1}} \right\rceil + \left\lceil \frac{{T_m}^{N_c-2} + {J_m}^{N_c-1}}{MUT_m{}^{N_c-1}} \right\rceil, & \text{if } {\xi_m}^{N_c-1} = \text{M} \end{cases} \tag{5.38}$$

In this case, $\Omega_c$ is equal to $\Omega_c^{N_c-1_2}$ because we consider only two lowest priority messages. It should be noted that we consider period or minimum update time of $m_m^{N_c-2}$ if it is periodic or sporadic. However, if $m_m^{N_c-2}$ is mixed then we select the maximum between its period and minimum update time in (5.38).

Let us consider three lowest priority messages in $CC_c$ denoted by $m_m^{N_c-1}$, $m_m^{N_c-2}$ and $m_m^{N_c-3}$. We denote the maximum number of instances of $m_m^{N_c-1}$ occupying the transmit buffers ahead of $m_m^{N_c-3}$ by $\Omega_c^{N_c-1_3}$. Similarly, the maximum number of instances of $m_m^{N_c-2}$ occupying the transmit buffers ahead of $m_m^{N_c-3}$ be denoted by $\Omega_c^{N_c-2_3}$. The calculations for $\Omega_c^{N_c-1_3}$ and $\Omega_c^{N_c-2_3}$ are adapted from (5.38) as follows.

$$\Omega_c^{N_c-1_3} = \begin{cases} \left\lceil \frac{{T_m}^{N_c-3}+{J_m}^{N_c-1}}{{T_m}^{N_c-1}} \right\rceil, & \text{if } {\xi_m}^{N_c-1} = \text{P} \\ \left\lceil \frac{{T_m}^{N_c-3}+{J_m}^{N_c-1}}{{MUT_m}^{N_c-1}} \right\rceil, & \text{if } {\xi_m}^{N_c-1} = \text{S} \\ \left\lceil \frac{{T_m}^{N_c-3}+{J_m}^{N_c-1}}{{T_m}^{N_c-1}} \right\rceil + \left\lceil \frac{{T_m}^{N_c-3}+{J_m}^{N_c-1}}{{MUT_m}^{N_c-1}} \right\rceil, & \text{if } {\xi_m}^{N_c-1} = \text{M} \end{cases} \tag{5.39}$$

$$\Omega_c^{N_c-2_3} = \begin{cases} \left\lceil \frac{{T_m}^{N_c-3}+{J_m}^{N_c-2}}{{T_m}^{N_c-2}} \right\rceil, & \text{if } {\xi_m}^{N_c-2} = \text{P} \\ \left\lceil \frac{{T_m}^{N_c-3}+{J_m}^{N_c-2}}{{MUT_m}^{N_c-2}} \right\rceil, & \text{if } {\xi_m}^{N_c-2} = \text{S} \\ \left\lceil \frac{{T_m}^{N_c-3}+{J_m}^{N_c-2}}{{T_m}^{N_c-2}} \right\rceil + \left\lceil \frac{{T_m}^{N_c-3}+{J_m}^{N_c-2}}{{MUT_m}^{N_c-2}} \right\rceil, & \text{if } {\xi_m}^{N_c-2} = \text{M} \end{cases} \tag{5.40}$$

In this case, $\Omega_c$ is equal to the sum of $\Omega_c^{N_c-1_3}$ and $\Omega_c^{N_c-2_3}$ as follows.

$$\Omega_c = \Omega_c^{N_c-1_3} + \Omega_c^{N_c-2_3} \tag{5.41}$$

Similarly, the maximum number of instances for any arbitrary number $Z$ of lower priority messages occupying transmit buffers in $CC_c$ can be calculated using the following equation. We assume $Z$ to be smaller than or equal to $N_c$.

$$\Omega_c = \Omega_c^{N_c-1_Z} + \Omega_c^{N_c-2_Z} + \Omega_c^{N_c-3_Z} + ... + \Omega_c^{N_c-(Z-1)_Z} \tag{5.42}$$

In this manner, we need to keep on calculating the number of instances of lower priority messages occupying transmit buffers in $CC_c$ until the value of $\Omega_c$ exceeds $Sizeof(K_c)$. The starting value for $Z$ is 2. Once we have reached this condition, the highest priority message in this set of low priority messages is designated as $m_{h_{K_c}}$.

**Case1: When Message Under Analysis is Free from Priority Inversion**

Let the message under analysis be $m_m$ and it belongs to the node $CC_c$. Once again, $m_m$ is treated differently in the extended RTA based on its transmission type. In this case, we consider that $m_m$ is free from priority inversion, i.e., its priority is smaller than or equal to the priority of $m_{h_{K_c}}$.

**Case 1(a): When ($m_m$) is periodic or sporadic**

Most of the equations to calculate response time of $m_m$ from Subsection 5.5.1 are applicable in this case. However, the only difference lies in the calculations for the queueing delay $\omega_m$ and the length of priority level-m busy period $t_m$. The calculations for $\omega_m$ should take into account two more elements.

1. The copying delay (from the message queue to the transmit buffer) denoted by $CT_m$ for every instance of $m_m$ in the priority level-m busy period.

2. Additional jitter of higher priority messages that is experienced by $m_m$.

Adding these elements to (5.8) and (5.9), $\omega_m$ can be calculated as follows.

$$\omega_m^{n+1}(q) = B_m + q_m C_m + (q_m + 1)CT_m + \sum_{\forall m_k \in hp(m_m)} I_k C_k \qquad (5.43)$$

$$I_k = \begin{cases} \left\lceil \frac{\omega_m^n(q_m) + \hat{J}_k + \tau_{bit}}{T_k} \right\rceil, & \text{if } \xi_k = \text{P} \\ \left\lceil \frac{\omega_m^n(q_m) + \hat{J}_k + \tau_{bit}}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{S} \\ \left\lceil \frac{\omega_m^n(q_m) + \hat{J}_k + \tau_{bit}}{T_k} \right\rceil + \left\lceil \frac{\omega_m^n(q_m) + \hat{J}_k + \tau_{bit}}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{M} \end{cases} \qquad (5.44)$$

The calculations for $t_m$ should take into account only one more element, i.e., the additional jitter of higher priority messages that is experienced by $m_m$. Adding it to (5.10) and (5.11), $t_m$ can be calculated as follows.

$$t_m^{n+1} = B_m + \sum_{\forall m_k \in hep(m_m)} I'_k C_k \tag{5.45}$$

$$I'_k = \begin{cases} \left\lceil \frac{t_m^n + \hat{J}_k}{T_k} \right\rceil, & \text{if } \xi_k = \text{P} \\ \left\lceil \frac{t_m^n + \hat{J}_k}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{S} \\ \left\lceil \frac{t_m^n + \hat{J}_k}{T_k} \right\rceil + \left\lceil \frac{t_m^n + \hat{J}_k}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{M} \end{cases} \tag{5.46}$$

In (7.25) and (5.46), $\hat{J}_k$ is calculated by replacing the index m with k in (5.35) and (5.36).

**Case 1(b): When ($m_m$) is mixed**

Similar to Case 1(a), most of the equations to calculate response time of $m_m$ from Subsection 5.5.2 are applicable in this case. The only difference lies in the calculations for $\omega_m$ and $t_m$. The same arguments from Case 1(a) hold for the calculations of $\omega_m$. In this case, $t_m$ can be calculated using (7.26) and (5.46). However, the queueing delay should be calculated separately for periodic ($m_{m_P}$) and sporadic ($m_{m_S}$) copies of $m_m$ by integrating $CT_m$ and $\hat{J}_m$ in (5.22), (5.23), (5.31) and (5.32) as follows.

$$\omega_{m_P}^{n+1}(q_{m_P}) = B_m + q_{m_P} C_m + (q_{m_P} + 1) CT_m + \sum_{\forall m_k \in hp(m_m)} I_{k_P} C_k + SI_{m_S}^P \tag{5.47}$$

$$\omega_{m_S}^{n+1}(q_{m_S}) = B_m + q_{m_S} C_m + (q_{m_S} + 1) CT_m + \sum_{\forall m_k \in hp(m_m)} I_{k_S} C_k + SI_{m_P}^S \tag{5.48}$$

$$I_{k_P} = \begin{cases} \left\lceil \frac{\omega^n_{m_P}(q_{m_P}) + \hat{J}_k + \tau_{bit}}{T_k} \right\rceil, & \text{if } \xi_k = \text{P} \\ \left\lceil \frac{\omega^n_{m_P}(q_{m_P}) + \hat{J}_k + \tau_{bit}}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{S} \\ \left\lceil \frac{\omega^n_{m_P}(q_{m_P}) + \hat{J}_k + \tau_{bit}}{T_k} \right\rceil + \left\lceil \frac{\omega^n_{m_P}(q_{m_P}) + \hat{J}_k + \tau_{bit}}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{M} \end{cases} \tag{5.49}$$

$$I_{k_S} = \begin{cases} \left\lceil \frac{\omega^n_{m_S}(q_{m_S}) + \hat{J}_k + \tau_{bit}}{T_k} \right\rceil, & \text{if } \xi_k = \text{P} \\ \left\lceil \frac{\omega^n_{m_S}(q_{m_S}) + \hat{J}_k + \tau_{bit}}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{S} \\ \left\lceil \frac{\omega^n_{m_S}(q_{m_S}) + \hat{J}_k + \tau_{bit}}{T_k} \right\rceil + \left\lceil \frac{\omega^n_{m_S}(q_{m_S}) + \hat{J}_k + \tau_{bit}}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{M} \end{cases} \tag{5.50}$$

In (5.49) and (5.50), $\hat{J}_k$ is calculated by replacing $_m$ with $_k$ in (5.35) and (5.36).

**Case2: When Message Under Analysis is Subjected to Priority Inversion**

In this case, we consider that $m_m$ can undergo priority inversion, i.e., its priority is greater than the priority of $m_{h_{K_c}}$.

**Case 2(a): When ($m_m$) is periodic or sporadic**

Most of the equations to calculate response time of $m_m$ from Subsection 5.5.1 are applicable in this case. However, the only difference lies in the calculations for the queueing delay $\omega_m$, blocking delay $B_m$, and the length of priority level-m busy period $t_m$. The calculations for $\omega_m$ should take into account three more elements.

1. The copying delay (from the message queue to the transmit buffer) denoted by $CT_m$ for every instance of $m_m$ in the priority level-m busy period.
2. Additional jitter of higher priority messages that is experienced by $m_m$.
3. Additional blocking delay as shown in (5.37).

Adding these elements to (5.8) and (5.9), $\omega_m$ can be calculated as follows.

$$\omega_m^{n+1}(q) = \hat{B}_m + q_m C_m + (q_m + 1)CT_m + \sum_{\forall m_k \in hp(m_m)} I_k C_k \quad (5.51)$$

It should be noted that $B_m$ is replaced with $\hat{B}_m$ which is calculated using (5.37). $I_k$ in (5.51) is calculated differently for different values of $\xi_k$ (k is the index of any higher priority message) using (7.25).

The value of priority level-m busy period $t_m$ is calculated similar to Case 1(a) in Subsection 5.5.1. However, the calculations for $t_m$ should take into account two more elements.

1. Additional jitter of higher priority messages that is experienced by $m_m$.
2. Additional blocking delay as shown in (5.37).

Adding these elements to (5.10) and (5.11), $t_m$ can be calculated as follows.

$$t_m^{n+1} = \hat{B}_m + \sum_{\forall m_k \in hep(m_m)} I'_k C_k \quad (5.52)$$

$I'_k$ in (5.52) is calculated differently for different values of $\xi_k$ (k is the index of any higher priority message) using (5.46).

**Case 2(b): When ($m_m$) is mixed**

Similar to Case 2(a), most of the equations to calculate response time of $m_m$ from Subsection 5.5.2 are applicable in this case. The only difference lies in the calculations for $\omega_m$, $B_m$ and $t_m$. In this case, $t_m$ can be calculated using (5.52) and (5.46). However, the queueing delay should be calculated separately for periodic ($m_{m_P}$) and sporadic ($m_{m_S}$) copies of $m_m$ by integrating $CT_m$, $\hat{B}_m$, $\hat{J}_m$ in (5.22) and (5.23).

$$\omega_{m_P}^{n+1}(q_{m_P}) = \hat{B}_m + q_{m_P} C_m + (q_{m_P} + 1)CT_m + \sum_{\forall m_k \in hp(m_m)} I_{k_P} C_k + SI_{m_S}^P \quad (5.53)$$

$$\omega_{m_S}^{n+1}(q_{m_S}) = \hat{B}_m + q_{m_S} C_m + (q_{m_S} + 1)CT_m + \sum_{\forall m_k \in hp(m_m)} I_{k_S} C_k + SI_{m_P}^S \quad (5.54)$$

Where $I_{k_P}$ and $I_{k_S}$ are calculated using (5.49) and (5.50) respectively.

## 5.7 Integrating the Effect of Non-abortable Transmit Buffers with the Extended Worst-case RTA for CAN

In this section, we integrate the effect of non-abortable transmit buffers in the CAN controllers with the extended RTA of CAN for periodic, sporadic and mixed messages. Basically, we extend the analysis from Section 5.5 by adapting the analysis in [28]. We assume that the CAN controllers do not support transmission abort requests. We use the same assumption about the number of transmit buffers in the CAN controllers that we used in Section 5.6. That is, the CAN controllers are assumed to implement limited number of transmit buffers (at least three).

### 5.7.1 Additional Delay and Jitter due to Priority Inversion

When CAN controllers do not support transmission abort requests, a higher priority message may suffer from priority inversion and this, in turn, adds extra delay to its response time [28]. Consider an example of three controllers $CC_c$, $CC_j$, $CC_k$ connected to a single CAN network in Figure 5.7. Let $m_1$, belonging to $CC_c$, be the highest priority message in the system. Assume that when $m_1$ is ready to be queued, all transmit buffers in $CC_c$ are occupied by lower priority messages which cannot be aborted because the controllers implement non-abortable transmit buffers. In addition, $m_1$ can be blocked by any lower priority message because the lower priority message already started its transmission. In this example $m_1$ is blocked by $m_5$ that belongs to node $CC_k$. Since all transmit buffers in $CC_c$ are full, $m_1$ has to wait in the message queue until one of the messages in $K_c$ is transmitted. Let $m_4$ be the highest priority message in $K_c$. $m_4$ can be interfered by higher priority messages ($m_2$ and $m_3$) belonging to other nodes. Hence, it can be seen that priority inversion takes place because $m_1$ cannot start its transmission before $m_4$ finishes its transmission while $m_4$ has to wait until messages $m_2$ and $m_3$ are transmitted. This adds an additional delay to the worst-case response time of $m_1$. Let this additional delay for $m_1$ be denoted by $AD_1$. In this example, $AD_1$ is the sum of the worst-case transmission times of $m_2$, $m_3$ and $m_4$. Generally, this additional delay is denoted by $AD_m^N$ for any message $m_m$. As we discussed

in Subsection 5.6.1, this additional delay appears as additional jitter of $m_m$ as seen by the lower priority messages. Let the additional jitter be denoted by $AJ_m^N$. Where $AJ$ stands for "Additional Jitter" while the superscript "N" stands for Non-abortable transmit buffer.

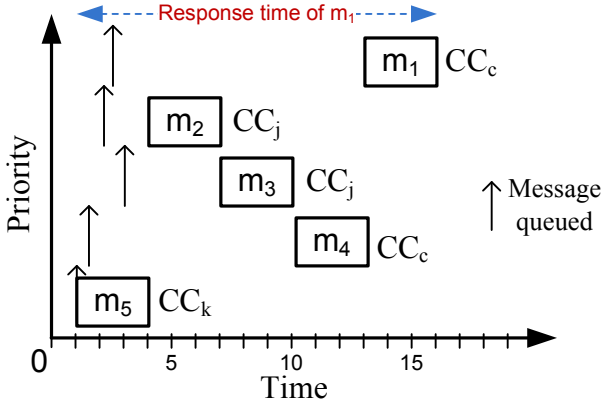


Figure 5.7: Demonstration of priority inversion in the case of non-abortable transmit buffers

### 5.7.2 Calculations for the Additional Delay, Jitter and Blocking

The calculations for the additional delay, additional jitter and extra blocking due to priority inversion (discussed in the above subsection) are adapted[3] from the existing analysis [28] to support mixed messages as well.

**Calculations for the additional delay.** Let $m_{h_{K_c}}$ be the highest priority message in the transmit buffers of $CC_c$ denoted by $K_c$. The calculations to determine the priority of $m_{h_{K_c}}$ can be adapted from Section 5.6.2. Let $m_m$ be the message under analysis whose priority is higher than $m_{h_{K_c}}$ and belongs to the same node $CC_c$. Assume all transmit buffers are occupied by lower priority messages when $m_m$ becomes ready for transmission. So $m_m$ has to wait until $m_{h_{K_c}}$ is transmitted. This waiting time for $m_m$ depends upon the response time of $m_{h_{K_c}}$. Let us term the response time of $m_{h_{K_c}}$ without its jitter as the modified response time and denote it by $R^*_{h_{K_c}}$. Mathematically,

[3] the existing analysis [28] does not support mixed messages

$$R^*_{h_{K_c}} = \omega^*_{h_{K_c}} + C_{h_{K_c}} \tag{5.55}$$

where, $C_{h_{K_c}}$ and $\omega^*_{h_{K_c}}$ denote the the worst-case transmission time and queueing delay of $m_{h_{K_c}}$ respectively. The reason for not considering jitter of $m_{h_{K_c}}$ as part of its modified response time is that $m_{h_{K_c}}$ is already in transmit buffer and hence its jitter will have no impact on the response time of $m_m$.

The message $m_{h_{K_c}}$ can be blocked by either one message in the set of lower priority messages belonging to other nodes or from its previous instance due to push-through blocking (discussed in Subsection 5.5.1). The queueing delay for $m_{h_{K_c}}$ is calculated as follows.

$$\omega^{*(n+1)}_{h_{K_c}} = B_{h_{K_c}} + \sum_{\forall m_k \in hp(m_{h_{K_c}})} I^*_k C_k \tag{5.56}$$

In (5.56), $I^*_k$ is calculated differently for different values of $\xi_k$ (k is the index of any higher priority message) as shown below.

$$I^*_k = \begin{cases} \left\lceil \frac{\omega^{*(n)}_{h_{K_c}} + \hat{J}_k + \tau_{bit}}{T_k} \right\rceil, & \text{if } \xi_k = \text{P} \\ \left\lceil \frac{\omega^{*(n)}_{h_{K_c}} + \hat{J}_k + \tau_{bit}}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{S} \\ \left\lceil \frac{\omega^{*(n)}_{h_{K_c}} + \hat{J}_k + \tau_{bit}}{T_k} \right\rceil + \left\lceil \frac{\omega^{*(n)}_{h_{K_c}} + \hat{J}_k + \tau_{bit}}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{M} \end{cases} \tag{5.57}$$

In (5.57), $\hat{J}_k$ is the additional jitter of higher priority message $m_k$ as seen by $m_{h_{K_c}}$. We will come back to its calculations later.

Once $m_{h_{K_c}}$ is in $K_c$, it cannot be interfered by $hp_c(m_{h_{K_c}})$ (i.e., the set of messages that belong to $CC_c$ and have priorities higher than the priority of $m_{h_{K_c}}$) because the buffers are non-abortable. Let this interference be denoted $IF^c_{h_{K_c}}$. However, the messages in $hp_c(m_{h_{K_c}})$ can indirectly interfere with $m_{h_{K_c}}$ before it occupies a buffer in $K_c$ by interfering with the messages in the set $hp(m_{h_{K_c}})$ belonging to other nodes. Let the interference received by $m_{h_{K_c}}$ from the messages in the set $hp(m_m)$ belonging to all nodes other than $CC_c$ be denoted by $IF^m_{h_{K_c}}$. The additional delay for $m_m$ will be equal to the difference between the modified response time $R^*_{h_{K_c}}$ of $m_{h_{K_c}}$ and the two combined interferences $IF^c_{h_{K_c}}$ and $IF^m_{h_{K_c}}$. It should be noted that $m_m$ can receive this additional delay from any message in node $CC_c$ whose priority is smaller

than $m_m$ and greater or equal to $m_{h_{K_c}}$. Hence, we need to calculate all these delays and select the maximum among them as the additional delay for $m_m$ as follows.

$$AD_m^N = \max_{\forall m_l \in CC_c \wedge (P_m < P_l \leq P_{h_{K_c}})} (R^*_{h_l} - IF^c_{h_{K_c}} - IF^m_{h_{K_c}}) \tag{5.58}$$

Where the interferences $IF^c_{h_{K_c}}$ and $IF^m_{h_{K_c}}$ are calculated as follows.

$$IF^c_{h_{K_c}} = \sum_{\forall m_i \in CC_c \wedge (1 \leq P_i < P_l)} I_{k_1} C_i \tag{5.59}$$

$$IF^m_{h_{K_c}} = \sum_{\forall m_j \notin CC_c \wedge (1 \leq P_j < P_m)} I_{k_2} C_j \tag{5.60}$$

In (5.59) and (5.60), the values for $I_{k_1}$ and $I_{k_2}$ are calculated differently for different values of $\xi_i$ and $\xi_j$ respectively as follows.

$$I_{k_1} = \begin{cases} \left\lceil \frac{R^*_{h_l} - C_l + \hat{J}_i + \tau_{bit}}{T_i} \right\rceil, & \text{if } \xi_i = \text{P} \\ \left\lceil \frac{R^*_{h_l} - C_l + \hat{J}_i + \tau_{bit}}{MUT_i} \right\rceil, & \text{if } \xi_i = \text{S} \\ \left\lceil \frac{R^*_{h_l} - C_l + \hat{J}_i + \tau_{bit}}{T_i} \right\rceil + \left\lceil \frac{R^*_{h_l} - C_l + \hat{J}_i + \tau_{bit}}{MUT_i} \right\rceil, & \text{if } \xi_i = \text{M} \end{cases} \tag{5.61}$$

$$I_{k_2} = \begin{cases} \left\lceil \frac{R^*_{h_l} - C_l + \hat{J}_j + \tau_{bit}}{T_j} \right\rceil, & \text{if } \xi_j = \text{P} \\ \left\lceil \frac{R^*_{h_l} - C_l + \hat{J}_j + \tau_{bit}}{MUT_j} \right\rceil, & \text{if } \xi_j = \text{S} \\ \left\lceil \frac{R^*_{h_l} - C_l + \hat{J}_j + \tau_{bit}}{T_j} \right\rceil + \left\lceil \frac{R^*_{h_l} - C_l + \hat{J}_j + \tau_{bit}}{MUT_j} \right\rceil, & \text{if } \xi_j = \text{M} \end{cases} \tag{5.62}$$

**Calculations for the additional jitter.** The total jitter of $m_m$ denoted by $\hat{J}_m$ as seen by the lower priority messages is the sum of its original jitter $J_m$ and the additional jitter due to priority inversion as follows.

$$\hat{J}_m = J_m + AJ_m^N \tag{5.63}$$

The additional jitter $AJ_m^N$ is calculated similar to the additional delay $AD_m^N$. However, we need to subtract only interference $IF_{h_{K_c}}^c$ from $R_{h_l}^*$ because $m_{h_l}$ cannot be interfered by higher priority messages from the same node after it has been transferred to the transmit buffer. Therefore, $AJ_m^N$ is calculated as follows:

$$AJ_m^N = \max_{\forall m_l \in CC_c \wedge (P_m < P_l \leq P_{h_{K_c}})} (R_{h_l}^* - IF_{h_{K_c}}^c) \tag{5.64}$$

Where, $IF_{h_{K_c}}^c$ is calculated using (5.59) and (5.61).

**Calculations for the blocking delay.** When $m_m$ is subjected to priority inversion due to non-abortable transmit buffers, it experiences an extra amount of blocking in addition to the original blocking delay $B_m$. The total blocking delay for $m_m$ denoted by $\hat{B}_m$ is the maximum value between the original blocking delay $B_m$ and additional delay $AD_m^N$. $B_m$ is calculated using (5.7) while $\hat{B}_m$ is calculated as follows.

$$\hat{B}_m = \max(B_m, AD_m^N) \tag{5.65}$$

It is important to note that equations (5.55), (5.58), and (5.63) are implicitly dependent on each other. Therefore, they are solved simultaneously and iteratively until two consecutive solutions of each equation become equal or the solutions exceed the message deadline in which case the message is deemed unschedulable.

### 5.7.3 Extended RTA

As discussed in the example given in Section 5.6.2, some messages will be safe from priority inversion, whereas other messages in the same node may suffer from priority inversion. Therefore, we consider two different cases for calculating response times of messages in CAN with non-abortable transmit buffers: Case(1) when message under analysis is free from priority inversion, and Case (2) when message under analysis is subjected to priority inversion. In each of these cases, we treat the message under analysis differently based on its transmission type: Case (a) when message under analysis is periodic or sporadic, and Case (b) when message under analysis is mixed. This is exactly similar to the extended analysis for periodic, sporadic and mixed messages in

CAN with abortable transmit buffers that is discussed in the previous section. All equations for the response-time calculations from (5.43) to (5.54) from the previous section are applicable with the following changes.

1. Since the controllers implement non-abortable transmit buffers, there will be no copying delays. Therefore, the copying delay denoted by $CT_m$ should be neglected. The following changes should be made in the analysis from the previous section:
   (a) $(q_m + 1)CT_m$ should be removed from equations (5.43) and (5.51),
   (b) $(q_{m_P} + 1)CT_m$ should be removed from equations (5.47) and (5.53),
   (c) $(q_{m_S} + 1)CT_m$ should be removed from equations (5.48) and (5.54).
2. The total jitter of $m_m$ denoted by $\hat{J}_m$ as seen by the lower priority messages should be calculated using (5.63) instead of (5.35).
3. Additional delay should be calculated using (5.58).
4. The total blocking delay for $m_m$ denoted by $\hat{B}_m$ should be calculated using (5.65) instead of (5.37).

## 5.8 Comparative Evaluation

In this section, we perform a number of tests on a message set consisting of 50 messages to evaluate and compare the three extended analyses with each other. The message set is generated using the NETCARBENCH tool [46] which is a benchmark for techniques and tools that are used in the design of automotive embedded systems. In all these tests, the system consists of 5 ECUs that are connected to a single CAN network. The speed of the network is set to 250 Kbit/s (kilo bits per second). The buffer limitations in the ECUs are different in different tests. Each message in the generated message set has a unique priority. The highest priority is 1, whereas the lowest priority is 50. It should be noted that the NETCARBENCH tool cannot generate mixed messages. After generating the message set from NETCARBENCH, we randomly assigned mixed, periodic, and sporadic transmission types to 40%, 30%, and 30% messages respectively. This means, there are 20 mixed, 15 periodic and 15 sporadic messages in the message set under analysis. The messages are equally distributed among the ECUs, i.e., each ECU sends 10 messages over the network out of which 4 are mixed, 3 are periodic, and 3 are sporadic.

### 5.8.1 Comparison of the Extended Analyses

In the first test, we consider three different cases: (i) all ECUs are assumed to have no buffer limitations in the CAN controllers, (ii) each ECU in the system implements three transmit buffers in the CAN controller and the buffers are abortable, and (iii) each ECU in the system implements three transmit buffers in the CAN controller and the buffers are non-abortable. In the case (i), we analyze the message set with the extended analysis that does not take into account buffer limitations in the CAN controllers (the analysis from Section 5.5). In the case (ii), we analyze the same message set with the extended analysis that considers abortable transmit buffers (the analysis from Section 5.6). Finally in the case (iii), we analyze the same message set with the extended analysis that considers non-abortable transmit buffers (the analysis from Section 5.7).

Figure 5.8(a) depicts the bar graph that shows the response times of messages that are calculated with three different analyses discussed above. The blue bars (first one in each set of the three bars) represent the response times of messages that are calculated using the analysis that does not consider buffer limitations. The red bars (second one in each set of the three bars) represent the response times of messages that are calculated using the analysis with abortable transmit buffers. Whereas, the green bars (third one in each set of the three bars) represent the response times of messages that are calculated using the analysis with non-abortable transmit buffers. In order to magnify the difference between the response times in different cases, we split the Figure 5.8(a) into five parts with each figure showing the set of response times for every 10 messages as shown in Figures 5.8(b), 5.8(c), 5.8(d), 5.8(e) and 5.8(f) respectively.

The results indicate that the message response times that are calculated with the extended analysis without buffer limitations is always smaller than or equal to the response times that are calculated with the extended analysis with buffer limitations. This means that if there are limited buffers in the CAN controllers and the effect of buffer limitations is not considered in the RTA, the calculated response times can be optimistic.

Let's compare the response times that are calculated with the extended analyses with buffer limitations, i.e., abortable and non-abortable transmit buffers separately. Apart from those lowest priority messages that are equal to the number of transmit buffers in each CAN controller (three lowest priority messages in this case), the response times of messages are smaller if CAN controllers implement abortable transmit buffers compared to non-abortable transmit buffers. On the other hand, the response times of the three lowest priority

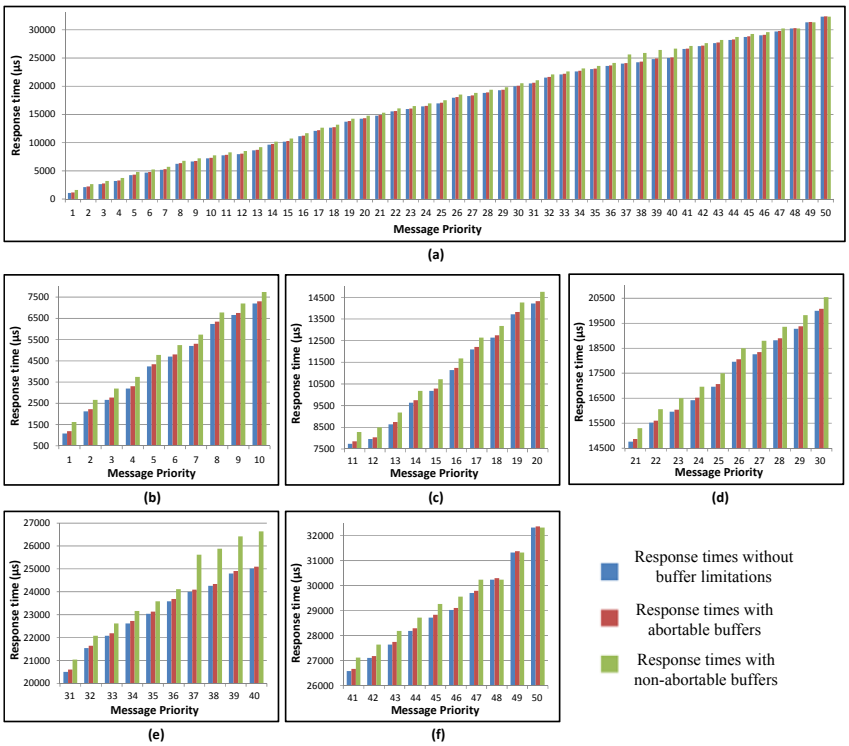


Figure 5.8: Comparison of message response times that are calculated with the extended analyses (i) without buffer limitations, (ii) with abortable transmit buffers, and (iii) with non-abortable transmit buffers.

messages in the system with non-abortable transmit buffers is smaller compared to the system with abortable transmit buffers because the three lowest priority messages are free from priority inversion (this was discussed in Section 5.7). In fact, the response times of the three lowest priority messages in the system with non-abortable transmit buffers match their response times when there are no buffer limitations in the CAN controllers. It can be concluded that it is more feasible to use CAN controllers with abortable transmit buffers compared to non-abortable transmit buffers. Moreover, it is important to use the RTA that matches the actual limitations and constraints in the hardware, device drivers and protocol stack.

### 5.8.2 Application of the Extended Analyses to Heterogeneous Systems

In the second test, we consider the case of a heterogeneous system in addition to the three cases from the first test. By heterogeneous system, we mean that the ECUs have different buffer limitations. That is, two ECUs implement abortable transmit buffers, two implement non-abortable transmit buffers while there are no buffer limitations in one ECU. Those ECUs that have buffer limitations implement three transmit buffers. We use the same message set in the heterogeneous system. In this case the messages that belong to the ECUs without buffer limitations are analyzed with the extended analysis from Section 5.5. The messages that belong to the ECUs that implement abortable transmit buffers are analyzed with the extended analysis from Section 5.6. Similarly, the messages that belong to the ECUs that implement non-abortable transmit buffers are analyzed with the extended analysis from Section 5.7.

In order to magnify the difference between the response times in different cases, we show the response time results for every 10 messages separately in Figures 5.9(a), 5.9(b), 5.9(c), 5.9(d) and 5.9(e) respectively. In each figure, the blue bars (first one in each set of the four bars) represent the response times of messages that are calculated using the analysis that does not consider buffer limitations. The red bars (second one in each set of the four bars) represent the response times of messages that are calculated using the analysis with abortable transmit buffers. Similarly, the green bars (third one in each set of the four bars) represent the response times of messages that are calculated using the analysis with non-abortable transmit buffers. Whereas, the purple bars (fourth one in each set of the four bars) represent the response times of messages in the heterogeneous system that are calculated using all three extended analyses together.

The results indicate that the message response times in the heterogeneous system are always greater than the message response times when the ECUs have no buffer limitations or the ECUs implement abortable transmit buffers. However, the response times of the 47 highest priority messages in the heterogeneous system are smaller than their response times when the ECUs implement non-abortable transmit buffers. Whereas, this trend is reversed for the three lowest priority messages because these messages are free from priority inversion.

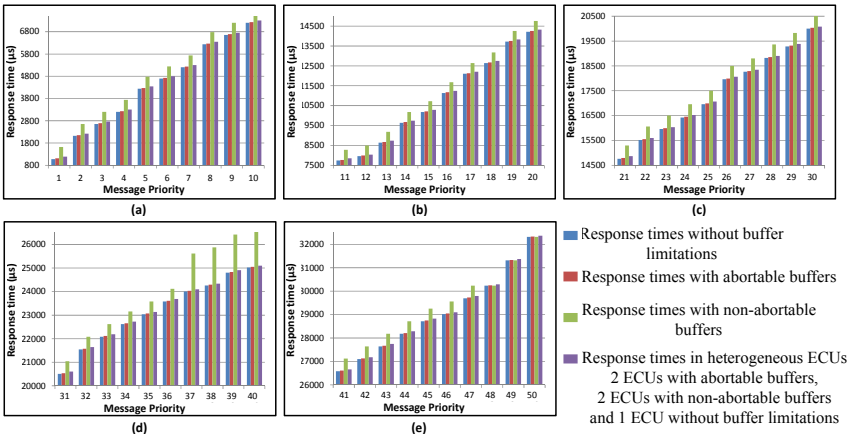


Figure 5.9: Comparison of message response times that are calculated with the extended analyses (i) without buffer limitations, (ii) with abortable transmit buffers, (iii) with non-abortable transmit buffers, and (iv) all three analysis in (i), (ii) and (iii) are applied on a heterogeneous system.

### 5.8.3 Effect of Copy Times of Messages on their Response Times

In the third test, we explore the effect of message copy times on their response times in the systems where ECUs implement three transmit buffers which are of abortable type. We use the same message set that we used in the previous tests. In this test, we consider six different cases with respect to the amount of message copy times: (i) copy time of all messages is four times the transmission time of a single bit of data over CAN (1-bit more time than the time required for inter-frame space of 3-bits), (ii) copy time of each message is 5% of its transmission time, (iii) copy time of each message is 10% of its transmission time, (iv) copy time of each message is 15% of its transmission time, (v) copy time of each message is 20% of its transmission time, and (vi) copy time of each message is 25% of its transmission time.

We analyze the message set in all these cases with the extended analysis from Section 5.6. In order to magnify the difference between the response times in different cases, we show the response time results for every 10 messages separately in Figures 5.10(a), 5.10(b), 5.10(c), 5.10(d) and 5.10(e) respectively. In each figure, the dark blue bars (first one in each set of the seven

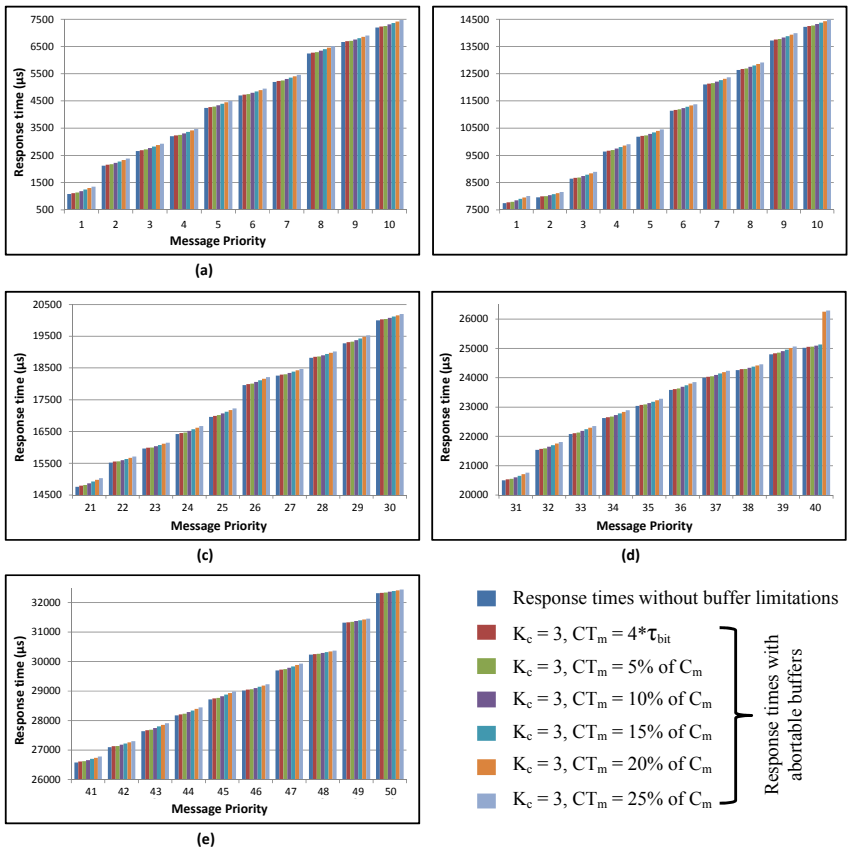


Figure 5.10: Comparison of message response times that are calculated with the extended analysis with abortable transmit buffers with different amount of message copy times.

bars) represent the response times of messages that are calculated using the analysis that does not consider buffer limitations. These response times are used just for the reference. The rest of the bars (in each set of the seven bars) represent the response times of messages that are calculated using the extended analysis with abortable transmit buffers while taking into account the different amount of message copy times as shown in Figure 5.10.

The results indicate that the increase in the response times of messages

is directly proportional to the increase in the amount of message copy times. If the message copy time is less than the inter-frame space (time required to transmit 3-bits of data on CAN), the response times of messages in the system with abortable transmit buffers becomes equals to the response times of same messages in the system with no buffer limitations.

### 5.8.4 Effect of the Number of Transmit Buffers on Message Response Times

In the fourth test, we explore the effect of the number of transmit buffers on message response times in the systems where ECUs implement non-abortable transmit buffers. Once again, the same message set is used. In this test, we consider eight different cases with respect to the number of transmit buffers in the CAN controllers, i.e., the number of transmit buffers in each ECU are equal to: (i) three, (ii) four, (iii) five, (iv) six, (v) seven, (vi) eight, (vii) nine, and (viii) ten. We analyze the message set in all these cases separately with the extended analysis from Section 5.7.

In order to magnify the difference between the response times in different cases, we show the response time results for every 10 messages separately in Figures 5.11(a), 5.11(b), 5.11(c), 5.11(d) and 5.11(e) respectively. In each figure, the dark blue bars (first one in each set of the nine bars) represent the response times of messages that are calculated using the extended analysis that does not consider buffer limitations. These response times are used just for the reference. The rest of the bars (in each set of the nine bars) represent the response times of messages that are calculated using the extended analysis with non-abortable transmit buffers while taking into account the different number of transmit buffers in the CAN controllers as shown in Figure 5.11.

As expected, the response times of messages in the system with no buffer limitations are always smaller than or equal to their response times when the ECUs in the system implement non-abortable transmit buffers. Let's consider the three lowest priority messages (priorities 48, 49 and 50). The response times of these messages are equal in all the cases because there are at least 3 transmit buffers in every ECU in each case. Therefore, these messages are free from priority inversion. Now consider the message with priority equal to 47. This message has the highest response time when ECUs contain 3 transmit buffers as shown by the red bar (second in the set of the nine bars) in Figure 5.11(e). Since, it is fourth lowest priority message in the system, it is not save from priority inversion when there are three transmit buffers in each ECU. Similarly, for the message with priority equal to 46, the message has higher

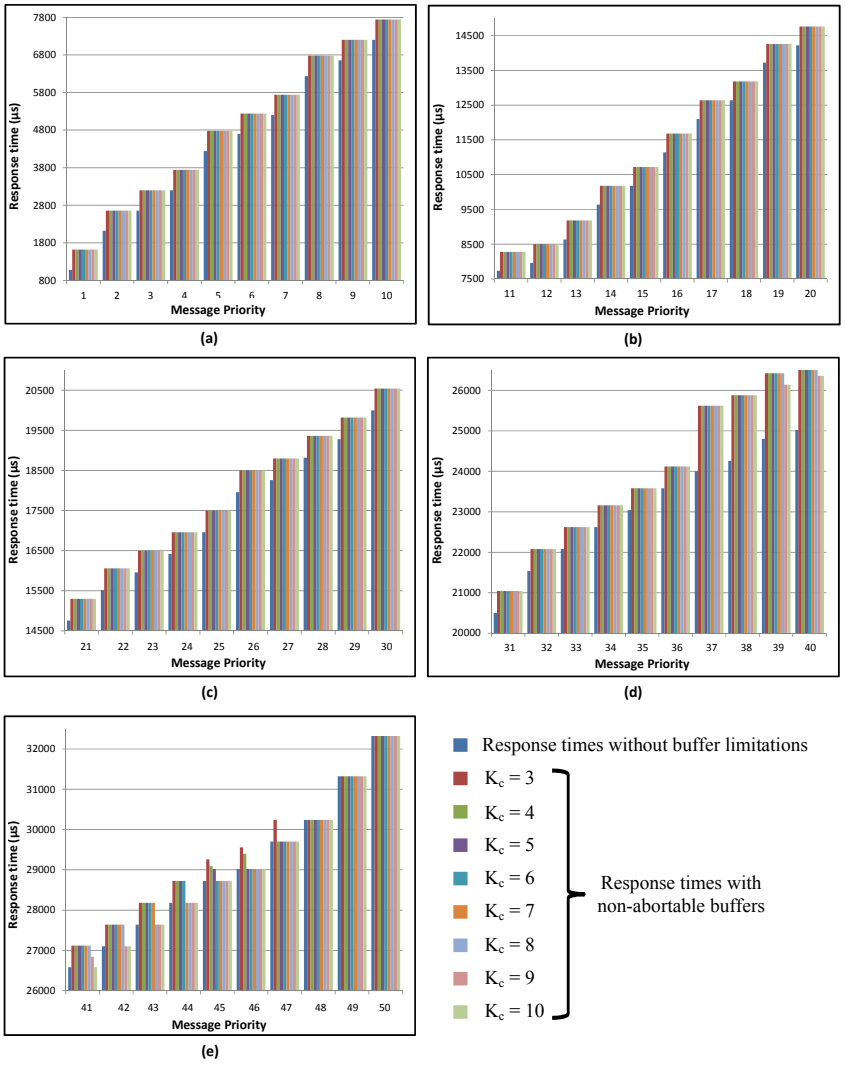


Figure 5.11: Comparison of message response times that are calculated with the extended analysis with non-abortable transmit buffers with different size of transmit buffers in the CAN controllers.

response times in the system where ECUs implement 3 and 4 transmit buffers as shown by red and green bars (second and third in the set of the nine bars) in Figure 5.11(e) respectively. This trend of the increase in the response times of messages with priorities 45, 44, 43, 42, 42, and 40 continues as the number of transmit buffers in the ECUs keeps on increasing from 5 to 10. However, this trend breaks at message with priority equal to 38 where it has equal response times in different systems with respect to different number of transmit buffers in the ECUs. This new trend remains the same for the rest of the highest priority messages (messages with priorities from 37 to 1).

## 5.9 Conclusion

The existing worst-case Response Time Analysis (RTA) for Controller Area Network (CAN) does not support mixed messages. Mixed messages can be queued for transmission both periodically and sporadically. They are implemented by some of the higher-level protocols and commercial extensions of CAN that are used in the automotive industry. We extended the existing analysis to support mixed messages. The extended analysis is able to calculate the upper bounds on the response times of CAN messages with all types of transmission patterns, i.e., periodic, sporadic and mixed. Furthermore, we integrated the effect of hardware and software limitations in the CAN controllers and device drivers such as abortable and non-abortable transmit buffers with the extended analysis for mixed messages. The extended analyses are also applicable to heterogeneous types of systems where ECUs (Electronic Control Units) are supplied by different tier-1 suppliers. These ECUs may have different limitations in the CAN controllers, device drivers and protocol stack.

We also conducted a case study to show the applicability of the extended analyses and performed the comparative evaluation of the extended analyses. The evaluation results indicate that if there are limited number of transmit buffers in the CAN controllers and the effect of buffer limitations is not considered in the RTA, the calculated response times can be optimistic. Hence, it is important to use the RTA that matches the actual limitations and constraints in the hardware, device drivers and protocol stack.

An interesting future work is to integrate the effect of buffer limitations with the offset-based RTA for mixed messages in CAN. Currently, we have implemented the extended analysis for mixed messages in CAN without buffer limitations in an existing industrial tool suite the Rubus-ICE [19]. In the future, we plan to implement the extended analysis for mixed messages with buffer

limitations in the Rubus-ICE.

## Acknowledgement

This work is supported by the 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 AB, BAE Systems Hägglunds and Volvo Construction Equipment (VCE), Sweden.

# Appendix A

| Notation | Explanation |
|---|---|
| $CC_n$ | CAN Controller (or a node) $_n$ |
| $\aleph$ | Total number of messages in the system |
| $\aleph_n$ | Set of messages belonging to $CC_n$ |
| $K_n$ | Transmit buffers in $CC_n$ |
| $m_n$ | Any message $_n$ |
| $ID_n$ | Unique identifier of $_n$ |
| $P_n$ | Priority of $_n$ |
| $hp(m_n)$ | Set of higher priority messages than $m_n$ |
| $lp(m_n)$ | Set of lower priority messages than $m_n$ |
| $hep(m_n)$ | Set of higher and equal priority messages than $m_n$ |
| $lep(m_n)$ | Set of lower and equal priority messages than $m_n$ |
| $FRAME\_TYPE$ | Specifies whether the frame is a standard or an extended CAN frame |
| $\xi_n$ | Transmission type of $m_n$. It specifies whether $m_n$ is periodic ($P$), sporadic ($S$) or mixed ($M$) |
| $C_n$ | Worst-case transmission time of $m_n$ |
| $J_n$ | Queueing jitter of $m_n$ |
| $s_n$ | Size of data payload in $m_n$ |
| $T_n$ | Transmission period of $m_n$ |
| $MUT_n$ | Minimum Update Time of $m_n$. It is the minimum time that should elapse between the transmission of any two sporadic messages |
| $B_n$ | Blocking time of $m_n$ |
| $R_n$ | Worst-case response time of $m_n$ |
| $D_n$ | Deadline of $m_n$ |
| $\omega_n$ | Queueing delay for $m_n$ |
| $\tau_{bit}$ | Time required to transmit a single bit of data over the CAN network |
| $t_n$ | Length of the priority level-n busy period |
| $U_m$ | Bus utilization for priority level-n |
| $Q_n$ | Number of instances of $m_n$ that are queued in priority level-n busy period |

Table 5.1: Notations and terminology

# Bibliography


[1] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Extending schedulability analysis of Controller Area Network (CAN) for mixed (periodic/sporadic) messages. In *16th IEEE Conference on Emerging Technologies and Factory Automation (ETFA), Sep., 2011*, Sep. 2011.

[2] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Response time analysis for mixed messages in CAN supporting transmission abort requests. In *7th IEEE International Symposium on Industrial Embedded Systems (SIES), June, 2012*, June 2012.

[3] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Extending response-time analysis of mixed messages in CAN with controllers implementing non-abortable transmit buffers. In *17th IEEE Conference on Emerging Technologies and Factory Automation (ETFA), Sep., 2012*, Sep. 2012.

[4] Robert Bosch GmbH. CAN specification version 2.0. 1991. Postfach 30 02 40, D-70442 Stuttgart.

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

[6] Automotive networks. CAN in Automation (CiA), 2011. http://www.can-cia.org/index.php?id=416, accessed on Feb. 05, 2014.

[7] Controller Area Network introduced 25 years ago, 2011. CAN Newsletter, March 2011. http://www.can-cia.org, accessed on Feb. 05, 2014.

[8] Marco Di Natale, Haibo Zeng, Paolo Giusto, Arkadeb Ghosal. *Understanding and Using the Controller Area Network Communication Protocol*. Springer, 2012.

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

[10] 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.

[11] 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.

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

[13] Wan-Chen Lu, Jen-Wei Hsieh, Wei-Kuan Shih and Tei-Wei Kuo. A faster exact schedulability analysis for fixed-priority scheduling. *Journal of Systems and Software*, 79(12):1744 – 1753, 2006.

[14] M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer Journal,*, 29(5):390–395, 1986.

[15] Mikael Nolin, Jukka Mäki-Turja, and Kaj Hänninen. Achieving industrial strength timing predictions of embedded system behavior. In *International Conference on Embedded Systems and Applications (ESA), 2008*, pages 173–178, 2008.

[16] 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.

[17] Volcano Network Architect. Mentor Graphics, http://www.mentor.com/products/vnd/communication-management/vna, accessed on Feb. 05, 2014.

[18] 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.

[19] Rubus-ICE: Integrated component Development Environment, http://www.arcticus-systems.com, accessed on feb. 05, 2014.

[20] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Support for end-to-end response-time and delay analysis in the industrial tool suite: Issues, experiences and a case study. *Computer Science and Information Systems, ISSN: 1361-1384,*, 10(1), 2013.

[21] D.A. Khan, R.J. Bril, and N. Navet. Integrating hardware limitations in CAN schedulability analysis. In *8th IEEE International Workshop on Factory Communication Systems (WFCS), May, 2010*, pages 207 –210, May 2010.

[22] Rob Davis, Steffen Kollmann, Victor Pollex, and Frank Slomka. Schedulability analysis for controller area network (CAN) with FIFO queues priority queues and gateways. *Real-Time Systems*, 49(1):73–116, 2013.

[23] Marco Di Natale and Haibo Zeng. Practical issues with the timing analysis of the Controller Area Network. In *18th IEEE Conference on Emerging Technologies and Factory Automation (ETFA), Sep., 2013*, Sep. 2013.

[24] R. Davis and N. Navet. Controller area network (CAN) schedulability analysis for messages with arbitrary deadlines in FIFO and work-conserving queues. In *9th IEEE International Workshop on Factory Communication Systems (WFCS), May, 2012*, pages 33 –42, May 2012.

[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, July, 2011*, July 2011.

[26] A. Meschi, M. Di Natale, and M. Spuri. Priority inversion at the network adapter when scheduling messages with earliest deadline techniques. In *Eighth Euromicro Workshop on Real-Time Systems, 1996*, pages 243–248, 1996.

[27] Marco Di Natale. Evaluating message transmission times in Controller Area Networks without buffer preemption. In *8th Brazilian Workshop on Real-Time Systems, 2006*, 2006.

[28] D.A. Khan, R.I. Davis, and N. Navet. Schedulability analysis of CAN with non-abortable transmission requests. In *16th IEEE Conference on Emerging Technologies Factory Automation (ETFA), Sep., 2011*, Sep. 2011.

[29] Transmit Cancellation in AUTOSAR Specification of CAN Driver, Rel. 4.1, Rev. 3, Ver. 4.3.0. March, 2014. http://www.autosar.org/download /R4.1/AUTOSAR_SWS_CANDriver.pdf, accessed on May 05, 2014.

[30] Patrick Yomsi, Dominique Bertrand, Nicolas Navet, and Robert Davis. Controller Area Network (CAN): Response time analysis with offsets. In *9th IEEE International Workshop on Factory Communication Systems (WFCS), May, 2012*, May 2012.

[31] Alexander Szakaly. Response time analysis with offsets for CAN. Master's thesis, Department of Computer Engineering, Chalmers University of Technology, Nov. 2003.

[32] Mathieu Grenier, Lionel Havet, and Nicolas Navet. Pushing the limits of CAN- scheduling frames with offsets provides a major performance boost. In *4th European Congress on Embedded Real Time Software (ERTS), 2008*, 2008.

[33] Mathieu Grenier, Lionel Havet, and Nicolas Navet. Automotive embedded systems handbook. In Nicolas Navet and Francoise Siminot-Lion, editors, *Chapter 14: Scheduling messages with offsets on Controller Area Network - a major performance boost*. CRC Press, 2009.

[34] Yang Chen, Ryo Kurachi, Hiroaki Takada, and Gang Zeng. Schedulability comparison for CAN message with offset: Priority queue versus FIFO queue. In *19th International Conference on Real-Time and Network Systems (RTNS), Sep., 2011*, pages 181–192, Sep. 2011.

[35] Lei Du and Guoqing Xu. Worst case response time analysis for CAN messages with offsets. In *IEEE International Conference on Vehicular Electronics and Safety (ICVES), Nov., 2009*, pages 41 –45, Nov. 2009.

[36] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst. System level performance analysis - the SymTA/S approach. *Computers and Digital Techniques,*, 152(2):148–166, March 2005.

[37] S. Mubeen, J. Mäki-Turja and M. Sjödin. Response-time analysis of mixed messages in Controller Area Network with priority- and FIFO-queued nodes. In *9th IEEE International Workshop on Factory Communication Systems (WFCS), May, 2012*, May 2012.

[38] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Worst-case response-time analysis for mixed messages with offsets in Controller Area Network. In *17th IEEE Conference on Emerging Technologies and Factory Automation (ETFA), Sep., 2012*, Sep. 2012.

[39] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Extending offset-based response-time analysis for mixed messages in Controller Area Network. In *18th IEEE Conference on Emerging Technologies and Factory Automation (ETFA), Sep., 2013*, Sep. 2013.

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

[41] CANopen Application Layer and Communication Profile. CiA Draft Standard 301. Ver. 4.02. Feb., 2002. http://www.can-cia.org/index.php?id=440, accessed on Feb. 05, 2014.

[42] Requirements on Communication, Rel. 4.1, Rev. 3, Ver. 3.3.1, March, 2014. www.autosar.org/download/R4.1/AUTOSAR_SRS_COM.pdf, accessed on May 05, 2014.

[43] AUTOSAR Techincal Overview, Version 2.2.2., Release 3.1, The AUTOSAR Consortium, Aug., 2008. http://autosar.org, accessed on Feb. 05, 2014.

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

[45] Ian Broster. *Flexibility in Dependable Real-time Communication*. PhD thesis, University of York, August 2003.

[46] Christelle Braun, Lionel Havet, and Nicolas Navet. NETCARBENCH: A benchmark for techniques and tools used in the design of automotive communication systems. In *7th IFAC International Conference on Fieldbuses & Networks in Industrial & Embedded Systems, Nov., 2007*, Nov. 2007.

# Chapter 6

# Paper C: Extending Worst-Case Response-Time Analysis for Mixed Messages in Controller Area Network with Priority and FIFO Queues

Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin
In Journal of Access, IEEE, vol 2, nr 1, 2014.

**Abstract**

The existing worst-case response-time analysis for Controller Area Network (CAN) with nodes implementing priority and First In First Out (FIFO) queues does not support mixed messages. It assumes that a message is queued for transmission either periodically or sporadically. However, a message can also be queued both periodically and sporadically using mixed transmission mode implemented by several higher-level protocols for CAN that are used in the automotive industry. We extend the existing analysis for CAN to support any higher-level protocol for CAN that uses periodic, sporadic, and mixed transmission of messages in the systems where some nodes implement priority queues while others implement FIFO queues. In order to provide a proof of concept, we implement the extended analysis in a free tool, conduct an automotive-application case study, and perform comparative evaluation of the extended analysis with the existing analysis.

## 6.1 Introduction

The Controller Area Network (CAN) [1] is a widely used real-time network protocol in the automotive domain. In 2003, the International Organization for Standardization (ISO) standardized CAN in ISO 11898-1 [2]. It is a multi-master, event-triggered, serial communication bus protocol supporting bus speeds of up to 1 Mbit/s. CAN with Flexible Data-rate (CAN FD) [3] is a new protocol based on CAN that can achieve bus speed of more than 1 Mbit/s. According to CAN in Automation (CiA) [4], the estimated number of CAN enabled controllers sold in 2011 are about 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 the automotive applications. For example, there can be as many as 20 CAN networks[1] used in a modern heavy truck, while the number of CAN messages transmitted over these networks can be over 6000 [5]. These facts and figures indicate the popularity of CAN in the automotive domain. It is also used in other domains such as industrial control, medical equipments, maritime electronics, production machinery, and many others. There are a number of higher-level protocols for CAN that are developed for many industrial applications such as CAN Application Layer (CAL) [6], CANopen [7], Hägglunds Controller Area Network (HCAN) [8], and CAN for Military Land Systems domain (MilCAN) [9].

CAN finds its applications in the systems that have real-time requirements. This means that the time for response to some stimulus is as crucial as logical correctness of the response. In other words, logically correct but late response may be considered as bad as logically incorrect response. Hence, the providers of these systems are required to ensure that the actions by the systems will be taken at times that are appropriate to their environment. In order to provide evidence that each action by the system will be provided in a timely manner, *a priori* analysis techniques, such as schedulability analysis [10, 11, 12], have been developed by the research community. Response-Time Analysis (RTA) [13, 10, 11, 12] 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. 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 [14].

[1] Since, CAN uses bus topology, we use the terms network and bus interchangeably throughout the paper.

### 6.1.1 Extended Version

This paper extends our previous work that was presented in the 9th IEEE International Workshop on Factory Communication Systems (WFCS 2012) [15]. The workshop paper presents the response-time analysis for mixed messages in CAN with FIFO queues. However, it lacks the calculations for maximum buffering time in the FIFO queues which is an important factor in the response-time calculations. Moreover, it does not evaluate and compare the extended analysis with the other related analyses. In the extended version of the paper, we generalize the analysis, by complementing it with the algorithm to calculate maximum buffering time in the FIFO queues. Moreover, we implement the extended analysis in a freely-available tool. We also show the applicability of the extended analysis by conducting an automotive-application case study. We also perform extensive evaluation of the extended analysis.

### 6.1.2 Related Works

Tindell et al. [16] developed the schedulability analysis for CAN. It has been implemented in the automotive industrial tools such as Volcano Network Architect (VNA) [17]. Davis et al. [18] found the analysis to be flawed in some cases. Accordingly, they revisited and revised the original analysis. The revised analysis is also implemented in the existing industrial tool suite Rubus-ICE [19, 20] which is used by several international companies.

The scheduling model used in [16, 18] assumes that the messages are queued for transmission either periodically or sporadically. These analyses do not support the response time calculations for mixed messages in CAN, i.e., the messages that are simultaneously time (periodic) and event triggered. Mixed messages are implemented by several higher-level protocols based on CAN that are used in the automotive industry. Mubeen et al. [21] extended the seminal analysis [16, 18] to support the worst-case response-time calculations for mixed messages in CAN.

However, the analyses in [16, 18, 21] assume that the device drivers in the CAN controllers implement priority-based queues. This means that the highest priority message at each node[2] enters into the bus arbitration. This assumption may become invalid when some controllers in the network implement FIFO queues. Some examples of the CAN controllers implementing FIFO queues are Infineon XC161CS, Microchip PIC32MX, Renesas R32C/160 and XILINX

[2]It should be noted that a node or ECU contains a CAN controller. We overload the terms node, processor, Electronic Control Unit (ECU), and CAN controller throughout the paper.

LogiCORE IP AXI Controller [22, 23, 24]. Davis et al. [25, 22] extended the analysis for CAN where some nodes implement priority queues while others implement FIFO queues.

In the works in [25, 22], the message deadlines are assumed to be smaller than or equal to the corresponding periods. In [26], Davis et al. lifted this assumption by supporting the analysis for CAN messages with arbitrary deadlines. Furthermore, they extended their previous works to support RTA for CAN with FIFO and work-conserving queues. However, the analyses for CAN with FIFO queues do not support mixed messages.

### 6.1.3 Paper Contributions and Motivation

We identified that the existing RTA for CAN with FIFO queues [25, 22, 26] does not support the analysis of common message transmission patterns, i.e., mixed messages. These type of messages are implemented by some higher-level protocols for CAN that are used in the automotive industry. Further, the existing analysis for mixed messages in CAN [21] does not support the analysis of the systems containing nodes that implement FIFO queues. We extend the existing analysis for CAN with FIFO queues [25, 22, 26] by integrating it with the analysis for mixed messages in CAN with priority queues [21]. Moreover, we generalize the extended analysis for CAN with FIFO queues by presenting the algorithm for the calculations of maximum buffering time in the FIFO queues. The relationship between the existing and extended analyses is shown in Figure 6.1.

The extended analysis does not put any restrictions on the message deadlines, i.e., the deadline of a message can be lower, equal, or higher than its transmission period. The extended analysis is able to calculate the worst-case response times of periodic, sporadic and mixed CAN messages in networks where some nodes implement priority queues while others implement FIFO queues. We also implement the extended analysis in a freely-available tool [27]. Furthermore, we show the applicability of the extended analysis by conducting the automotive-application case study. We also perform extensive evaluation of the extended analysis.

The motivation for this work comes from the industrial requirements and the activity of implementing the holistic response-time analysis [28] in the existing industrial tool suite, Rubus-ICE [20]. This tool provides a model- and component-based development environment for resource-constrained automotive distributed real-time systems while supporting several higher-level protocols based on CAN.

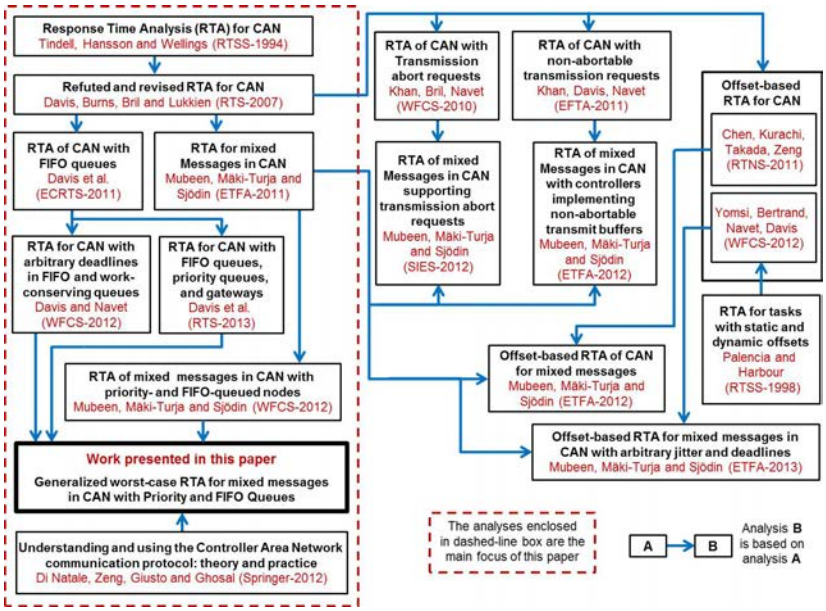


Figure 6.1: Relationship between the existing and extended analyses for CAN.

### 6.1.4 Paper Layout

The rest of the paper is organized as follows. In Section 6.2, we discuss mixed transmission patterns supported by several higher-level protocols for CAN. In Section 6.3, we discuss some common queueing policies in the transmit buffers of the CAN controllers. In Section 6.4, we describe the system model. In Section 6.5, we extend the existing analysis. Section 6.6 presents the case study and evaluation of the extended analysis. Finally, Section 6.7 summarizes and concludes the paper.

## 6.2 Mixed Transmission Patterns Supported by the Higher-level Protocols for CAN

In order to be consistent throughout the paper, we use the terms message and frame interchangeably. This is because we only consider messages that fit into one frame, i.e., the maximum size of a message can be 8 bytes. If a mes-

sage is queued for transmission at periodic intervals, we use the term "Period" to refer to its periodicity. On the other hand, a sporadic message is queued for transmission as soon as a sporadic event occurs that changes the value of one or more signals contained in the message provided the Minimum Update Time ($MUT$[3]) between the queueing of two successive sporadic messages has elapsed. The seminal RTA for CAN [16] and most of its extensions assume that the tasks queueing CAN messages are invoked either periodically or sporadically. However, there are some higher-level protocols and commercial extensions of CAN in which the tasks that queue the messages can be invoked periodically as well as sporadically. If a message is queued for transmission periodically as well as sporadically, the transmission type of a message is called mixed. That is, a mixed message is simultaneously time- and event-triggered. We identify three different types of implementations of the mixed messages by the higher-level protocols for CAN that are used in the automotive industry.

### 6.2.1 Implementation of Mixed Message in the CANopen Protocol

The CANopen protocol [7] supports mixed transmission that corresponds to the Asynchronous Transmission Mode coupled with Event Timer. The Event Timer is used for cyclic transmission of an asynchronous message. The mixed message in this protocol can be queued for transmission at the arrival of a sporadic event provided the Inhibit Time has expired. The Inhibit Time is the minimum time that must be allowed to elapse between the queueing of two consecutive messages. The mixed message can also be queued periodically when the Event Timer expires. The Event Timer is reset every time the message is queued. Once the mixed message is queued, any additional queueing of this message will not take place during the Inhibit Time [7]. The transmission pattern of the mixed message in the CANopen protocol is illustrated in Figure 6.2(a). The down-pointing arrows show queueing of the message while the numbers below them represent the instance number of the queued message. The upward lines labeled with alphabetic characters represent the arrival of events. Instance $1$ of the mixed message is queued as soon as the event $A$ arrives. Both the Event Timer and Inhibit Time are reset. As soon as the Event Timer expires, instance $2$ is queued due to periodicity and both the Event Timer and Inhibit Time are reset again. Instance $3$ of the mixed message is immediately queued upon arrival of the event $B$ because the Inhibit Time has

[3]We overload the term $MUT$ to refer to the *Inhibit Time* in the CANopen protocol [7] and the *Minimum Delay Time (MDT)* in the AUTOSAR communication [29].

already expired. Note that the Event Timer is also reset at the same time when instance 3 is queued as shown in Figure 6.2(a). The instance 4 of the mixed message is queued because of the expiry of the Event Timer. There exists a dependency relationship between the Inhibit Time and the Event Timer, i.e., the Event Timer is reset with every sporadic transmission.

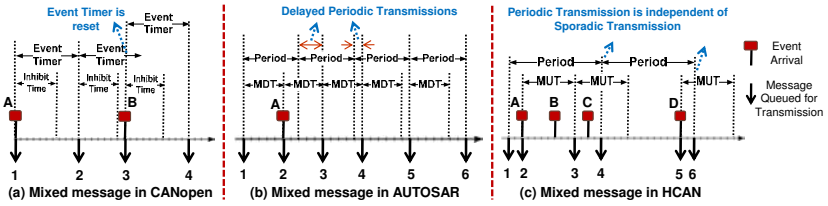


Figure 6.2: Mixed transmission pattern in higher-level protocols for CAN.

### 6.2.2 Implementation of Mixed Message in the AUTOSAR Communications

AUTOSAR (AUTomotive Open System ARchitecture) [30] can be viewed as a higher-level protocol if it uses CAN for network communication. Mixed transmission in AUTOSAR is widely used in practice. In this protocol, a mixed message can be queued for transmission periodically with the mixed transmission mode time period. The mixed message can also be queued at the arrival of an event provided the Minimum Delay Time ($MDT$) has been expired. However, each transmission of the mixed message, regardless of being periodic or sporadic, is limited by the $MDT$ timer. This means that both periodic and sporadic transmissions will always be delayed until the expiry of the $MDT$ timer. Figure 6.2(b) shows the transmission pattern of the mixed message implemented by AUTOSAR. The $MDT$ timer is started as soon as the first instance of the mixed message is queued due to partly periodic nature of the mixed message. Its second instance is queued immediately upon arrival of the event $A$ because the $MDT$ timer has already expired. The next periodic transmission is scheduled 2 time units after the transmission of instance 2. However, the next two periodic transmissions corresponding to instances 3 and 4 are delayed because the $MDT$ timer is still running. The transmissions that are delayed due to non-expiry of the $MDT$ timer are identified in Figure 6.2(b). The periodic transmissions corresponding to instances 5 and 6 take place at the scheduled times because the $MDT$ timer is already expired in both cases.

### 6.2.3 Implementation of Mixed Message in the HCAN Protocol

The mixed message in the HCAN protocol [8] contains signals out of which some are periodic and some are sporadic. The 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 the $MUT$ between the queueing of two successive sporadic instances of the mixed message has elapsed. Hence, the transmission of the mixed message due to arrival of events is constrained by the $MUT$. The transmission pattern of mixed message in the HCAN protocol is illustrated in Figure 6.2(c). Instance 1 of the mixed message is queued because of periodicity. As soon as event $A$ arrives, instance 2 is queued. When event $B$ arrives, the next instance of the mixed message is not queued immediately because the $MUT$ is not expired yet. As soon as the $MUT$ expires, the third instance is queued. The third instance contains the signal changes that correspond to event $B$. Similarly, the next instance of the mixed message is not immediately queued when the event $C$ arrives because the $MUT$ is not expired. Instance 4 of the mixed message is queued because of periodicity. Although, the $MUT$ was not expired, the event signal corresponding to event $C$ was packed in instance 4 and queued as part of the periodic message. Hence, there is no need to queue an additional sporadic instance of the mixed message when the $MUT$ expires. This indicates that the periodic transmission of a mixed message cannot be interfered by its sporadic transmission. This is a unique property of the HCAN protocol. When the event $D$ arrives, a sporadic instance of the mixed message is immediately queued as message 5 because the $MUT$ has already expired. Instance 6 is queued due to partly periodic nature of the mixed message.

### 6.2.4 Comparison of the Three Implementations of Mixed Message

In the first implementation method, the Event Timer is reset every time the mixed message is queued for transmission. The implementation of the mixed message in method 2 is similar to method 1 to some extent. The main difference is that the periodic transmission can be delayed until the expiry of the $MDT$ in method 2. Whereas in method 1, the periodic transmission is not delayed, in fact, the Event Timer is restarted with every sporadic transmission. The $MDT$ timer is started with every periodic or sporadic transmission of the mixed message. Hence, the worst-case periodicity of the mixed

message in methods 1 and 2 can never be higher than the Inhibit Timer and the $MDT$ respectively. Therefore, the existing analyses for CAN with FIFO queues [25, 22, 26] hold intact. However, the periodic transmission is independent of the sporadic transmission in the third implementation method. The periodic timer is not reset with every sporadic transmission. The mixed message can be queued for transmission even if the $MUT$ is not expired. The worst-case periodicity of the mixed message is neither bounded by the period nor by the $MUT$. Therefore, the existing analyses for CAN with FIFO queues [25, 22, 26] cannot be applied to the mixed messages in the third implementation method.

## 6.3 Common Queueing Policies Used in the CAN Controllers

The timing behavior of CAN messages is influenced by many factors including the type of queueing polices implemented by the CAN device drivers and communication stack. The most common queueing policies in the nodes connected to the CAN network are priority- and FIFO-ordered policies.

### 6.3.1 Priority-ordered Queues

The CAN protocol implements priority-based arbitration for the transmission of messages on the network. This means, each node selects the highest priority message from its transmit buffers while entering into the bus arbitration. The highest priority message among the messages selected from each node wins the arbitration, i.e., the right to transmit over the network. Intuitively, the most natural queueing policy suited to CAN controllers is priority-ordered queueing.

Let us consider an example to demonstrate the priority-based queueing policy as shown in Figure 6.3. Let there be three nodes namely Node A, Node B and Node C that are connected to a single CAN network . Each node sends three messages over the network. Node A sends the messages $m_1$, $m_3$ and $m_5$; Node B sends the messages $m_2$, $m_4$ and $m_9$; whereas, Node C sends the messages $m_6$, $m_7$ and $m_8$. The subscript in the name of a message represents its priority. We assume that the smaller the value of the subscript, the higher the priority of the message. Intuitively, $m_1$ is the highest priority message, whereas, $m_9$ is the lowest priority message in the system.

In order to simplify the example, assume that the transmission periods of all messages are very high compared to their transmission times. Assume that

all messages in each node are queued for transmission. We also assume that there cannot be multiple instances of a message queued for transmission at the same time.

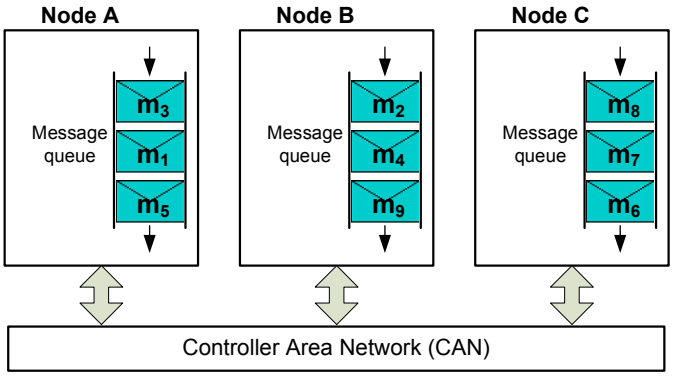


Figure 6.3: Example to demonstrate the effect of queueing policy on message transmission.

Let the nodes implement priority queues. Each node selects the highest priority message from its queue to enter into bus arbitration. In the first round, Nodes A, B, and C select messages $m_1$, $m_2$ and $m_6$ respectively. Message $m_1$ wins the arbitration and is transmitted over the network as shown in Figure 6.4. In the second round, Nodes A, B, and C pick messages $m_3$, $m_2$ and $m_6$ respectively. This time, message $m_2$ wins the arbitration and is transmitted over the network. Similar priority-based selection and arbitration continue during the rest of the rounds as shown in Figure 6.4.

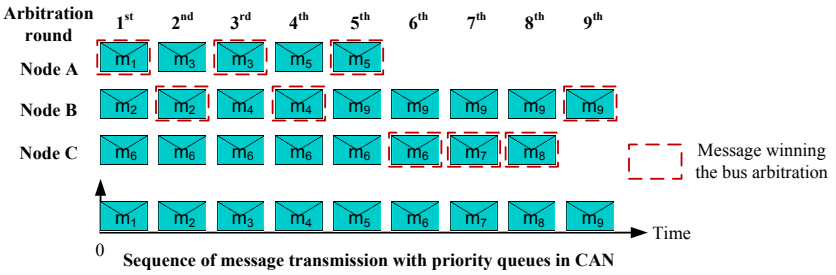


Figure 6.4: Demonstration of CAN arbitration and priority-based queueing.

### 6.3.2 FIFO Queues

The main advantages of FIFO queueing policy is that it is simple to implement and use. Some examples of the CAN controllers that implement FIFO queueing policy are Microchip PIC32MX, Infineon XC161CS, Renesas R32C/160 and XILINX LogiCORE IP AXI Controller [22, 23]. When nodes implement FIFO queues, the oldest message in the transmit queue of each node competes for the network with the oldest messages in the transmit queues in the rest of the nodes. It should be noted that even in the case of FIFO queues, the bus arbitration among CAN messages from different nodes is done on priority basis. Let us consider the three nodes, shown in Figure 6.3, implement FIFO queues. Intuitively, each node selects the oldest message in its queue to enter into the bus arbitration. In the first round, Nodes A, B, and C pick messages $m_5$, $m_9$ and $m_6$ respectively. Due to higher priority, message $m_5$ wins the arbitration and is transmitted over the network as shown in Figure 6.5. In the second round, Nodes A, B, and C pick messages $m_1$, $m_9$ and $m_6$ respectively. In this round, message $m_1$ wins the arbitration and is transmitted over the network. Similar FIFO selection and priority-based arbitration occur during the rest of the rounds as shown in Figure 6.5.

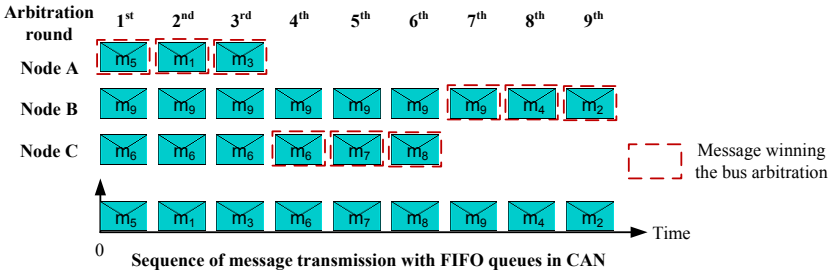


Figure 6.5: CAN arbitration and FIFO-based queueing.

### 6.3.3 Effect of Queueing Policy on the Response Times of Messages

When FIFO queues are used, the priorities of messages are often not respected in the transmit queue within a node, e.g., the lower priority message $m_5$ is transmitted before the highest priority message $m_1$ as shown in Figure 6.5. As a result, priority inversion can occur due to which higher priority messages

may have very large response times. This becomes evident by comparing the response time of message $m_2$ in the systems with priority and FIFO queues as shown in Figure 6.4 and Figure 6.5 respectively.

## 6.4 System Model

The system scheduling model is based on the seminal model in [16] and its extensions for FIFO queues [25] and mixed messages [21]. The system consists of a number of nodes connected to a single CAN network. A node may implement a priority queue or a FIFO queue. In the former case, the node is designated as a PQ-node and it enters the highest priority message from its transmit queue in the bus arbitration. Whereas, in the later case the node is identified as an FQ-node and it enters the oldest message from its transmit queue in the bus arbitration.

Each CAN message $m_m$ has a unique identifier and a priority denoted by $ID_m$ and $P_m$ respectively. The priority of a message is assumed to be equal to its ID. The priority of the message $m_m$ is considered higher than the priority of another message $m_n$ if $P_m < P_n$. Let the sets $hp(m_m)$, $lp(m_m)$, and $hep(m_m)$ contain the messages with priorities higher, lower, and equal and higher than $m_m$ respectively. Although the priorities of CAN messages are unique, the set $hep(m_m)$ is used in the case of mixed messages.

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 the standard CAN frame uses an 11-bit identifier whereas the extended CAN frame uses a 29-bit identifier. In order to keep the notations simple and consistent, we define a function $\xi_m$ that denotes the transmission type of a message. $\xi_m$ specifies whether $m_m$ is periodic ($P$), sporadic ($S$) or mixed ($M$). Formally, the domain of $\xi_m$ can be defined as follows.

$$\xi_m \in [P, \quad S, \quad M]$$

Each message $m_m$ has a transmission time $C_m$ and queueing jitter $J_m$ which is inherited from the task that queues $m_m$, i.e., the sending task. We assume that $J_m$ can be smaller, equal or greater than $T_m$ or $MUT_m$. Each message can carry a data payload that ranges from 0 to 8 bytes. This integer value is specified in the header field of the frame called Data Length Code and is denoted by $s_m$. In the case of periodic transmission, $m_m$ has a transmission period which is denoted by $T_m$. Whereas, in the case of sporadic transmission,

$m_m$ has the $MUT_m$ time. $B_m$ denotes the blocking time of $m_m$ which refers to the largest amount of time $m_m$ has to wait for the transmission of a lower priority message.

If an FQ-node transmits the message $m_m$ then the set of all messages transmitted by this node is defined by $M(m_m)$. The Lowest priority message in $M(m_m)$ is denoted by $L_m$. The sum of the transmission time of all the messages in $M(m_m)$ is identified by $C_m^{SUM}$. The transmission time of the shortest and longest messages in $M(m_m)$ are denoted by $C_m^{MIN}$ and $C_m^{MAX}$ respectively. $f_m$ denotes the maximum buffering time between the instant the message $m_m$ enters the FIFO queue and the instant it becomes the oldest message in the queue. It is equal to zero for a message belonging to a node that implements a priority queue [25].

We duplicate a message when its transmission type is mixed. Hence, each mixed message $m_m$ is treated as two separate messages, i.e., one periodic and the other sporadic. The duplicates share all the attributes except for $T_m$ and $MUT_m$. The periodic copy inherits $T_m$ while the sporadic copy inherits the $MUT_m$. 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). $m_m$ is deemed schedulable if its $R_m$ is less than or equal to its deadline $D_m$. The system is considered schedulable if all of its messages are schedulable.

We consider the deadlines to be arbitrary which means that they can be greater than the periods or $MUT$s of corresponding messages. We assume that the CAN controllers are capable of buffering more than one instance of a message. The instances of a message are assumed to be transmitted in the same order in which they are queued (i.e., we assume FIFO policy among the instances of the same message). For better readability, all the notations used in this paper are tabulated at the end of the paper.

## 6.5 Extended Analysis

We extend the existing analysis of CAN with both PQ-nodes and FQ-nodes [25] by adapting the RTA of CAN for mixed messages [21]. Let the message under analysis be denoted by $m_m$. The extended analysis treats a message differently based on its transmission type. Here we consider two different cases. In the first case, $m_m$ is assumed to be a periodic or a sporadic message. Whereas, $m_m$ is considered to be a mixed message in the second case.

### 6.5.1 Case 1: When $m_m$ is a Periodic or a Sporadic Message

Consider $m_m$ to be a periodic or a sporadic message. We calculate the worst-case response time of a message differently depending upon the type of the queueing policy implemented in the sending node. That is, we treat the message under analysis differently for the PQ-and FQ-nodes. Therefore, once again, we consider two cases: (a) the first case assumes that $m$ belongs to a node that implements priority queue, (b) the second case considers that $m$ belongs to a node that implements FIFO queue.

#### Case 1 (a): When $m_m$ Belongs to a Priority-queued Node

Since we consider arbitrary deadlines for messages, there can be more than one instance of $m_m$ that may become ready for transmission before the end of priority level-m *maximum busy period*. The maximum busy period is the longest contiguous interval of time during which $m_m$ is unable to complete its transmission due to two reasons. First, the network is occupied by the higher priority messages. In other words, at least one message of priority level-m or higher has not completed its transmission. Second, a lower priority message already started its transmission when $m_m$ is queued for transmission. The maximum busy period starts at the so-called *critical instant*. In a system where messages are scheduled without offsets, the critical instant corresponds to the point in time when all higher priority messages in the system are queued simultaneously with $m_m$ while their subsequent instances are queued after the shortest possible interval of time [18].

There can be another reason to check if more than one instance of $m_m$ is queued for transmission in the priority level-m maximum busy period. Since, the message transmission in CAN is non-preemptive, the transmission of previous instance of $m_m$ could delay the current instance of a higher priority message that may add to the interference received by the current instance of $m_m$. This phenomenon was identified by Davis et al. [18] and termed as "push-through interference". Because of this interference, a higher priority message may be waiting for its transmission before the transmission of the current instance of $m_m$ finishes. Hence, the length of busy period may extend beyond $T_m$ or $MUT_m$.

Intuitively, the response time of each instance of $m_m$ within priority level-m maximum busy period should be calculated. The largest value among the calculated response times of all instances of $m_m$ is considered as the worst-case response-time of $m_m$. Let $q_m$ be the index variable to denote instances of

$m_m$. The worst-case response time of $m_m$ is given by:

$$R_m = max\{R_m(q_m)\} \tag{6.1}$$

**Constituents of the worst-case response time.** According to the existing analysis [16, 18], the worst-case response-time of any instance of $m_m$ consists of three parts as follows.

1. The queueing jitter denoted by $J_m$. It is inherited from the sending task, i.e., the task that queues $m_m$ for transmission. Basically, it represents the maximum variation in time between the release of the sending task and queuing of the message in the transmit queue (buffers). It is calculated by taking the difference between the worst- and best-case response time of the sending task.

2. The worst-case transmission time denoted by $C_m$. It represents the longest time it takes for $m_m$ to be transmitted over the network.

3. The queueing delay denoted by $\omega_m$. It is equal to the longest time that elapses between the instant $m_m$ is queued by the sending task in the transmit queue and the instant when $m_m$ is about to start its successful transmission. In other words, $\omega_m$ is the interference caused by other messages to $m_m$.

Thus, the worst-case response time of any instance $q_m$ of a periodic or sporadic message $m_m$ is given by the following set of equations.

$$R_m(q_m) = \begin{cases} J_m + \omega_m(q_m) - q_m T_m + C_m, & \text{if } \xi_m = \text{P} \\ J_m + \omega_m(q_m) - q_m MUT_m + C_m, & \text{if } \xi_m = \text{S} \end{cases} \tag{6.2}$$

The terms $q_m\, T_m$ and $q_m MUT_m$ in (6.2) are used to support the response-time calculations for multiple instances of $m_m$. If the transmission type of $m_m$ is periodic then the message period is taken into account. However, if the transmission type of $m_m$ is sporadic, minimum update time is used in the above equation.

**Calculations for the worst-case transmission time $C_m$.** The worst-case transmission time of $m_m$ can be calculated using the method derived in [16] and later adapted in [18]. For the standard CAN identifier format, $C_m$ is calculated as follows.

$$C_m = \left(47 + 8s_m + \left\lfloor \frac{34 + 8s_m - 1}{4} \right\rfloor\right)\tau_{bit} \tag{6.3}$$

Where $\tau_{bit}$ represents the time required to transmit a single bit of data on the CAN network. Its value depends upon the speed of the network. In (6.3), $47$ is the number of bits due to protocol overhead. It is composed of start of frame bit (1-bit), arbitration field (12-bits), control field (6-bits), Cyclic Redundancy Check (CRC) field (16-bits), acknowledgement (ACK) field (2-bits), End of Frame (EoF) field (7-bits), and inter-frame space (3-bits). The number of bits due to protocol overhead in the case of extended CAN frame format is equal to $67$.

In [31], Broster identified that the analysis in [16, 18] uses $47$-bits instead of $44$-bits as the protocol overhead for a standard CAN identifier frame format. This is because the analysis in [16, 18] accounts 3-bit inter-frame space as part of the CAN frame. The 3-bit inter-frame space must be considered when calculating the interferences or blocking from other messages. However, Broster argued that this adds slight amount of pessimism to the response time of the message under analysis if the 3-bit inter-frame space is also considered in its transmission time. This is because the destination node can access the message before the inter-frame space. In order to avoid this pessimism, we subtract 3-bit time from the response time of the instance of the message under analysis.

The term $\left\lfloor \frac{34+8s_m-1}{4} \right\rfloor$ in (6.3) is added to compensate for the extra time due to *bit stuffing*. It should be noted that the bit sequences $000000$ and $111111$ are used for error signals in CAN. In order to be unambiguous in non-erroneous transmission, a stuff bit of opposite polarity is added whenever there are five bits of the same polarity in the sequence of bits to be transmitted [18]. The value $34$ indicates that only 34-bits out of 47-bits protocol overhead are subjected to bit stuffing. The term $\lfloor \frac{a}{b} \rfloor$ is the notation for *floor* function. It returns the largest integer that is less than or equal to $\frac{a}{b}$.

For the message with extended CAN identifier format, $C_m$ is calculated as follows.

$$C_m = \left(67 + 8s_m + \left\lfloor \frac{54 + 8s_m - 1}{4} \right\rfloor\right)\tau_{bit} \tag{6.4}$$

The calculations for $C_m$ in (6.3) can be simplified as follows.

$$C_m = (55 + 10s_m)\tau_{bit} \tag{6.5}$$

Similarly, the calculations for $C_m$ in (6.4) can be simplified as follows.

$$C_m = (80 + 10s_m)\tau_{bit} \tag{6.6}$$

**Calculations for the worst-case queueing delay $\omega_m$.** The calculations for $\omega_m$ should include the interference caused by all the other periodic, sporadic and mixed messages. The existing analyses for CAN with FIFO queues [25, 22, 26] have a limitation that they consider the effect of interference from only periodic and sporadic messages.

It is important to mention that CAN uses fixed-priority non-preemptive scheduling, 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 $m_m$ has to wait in the transmit queue because the higher priority messages win the arbitration, i.e., the right to transmit before $m_m$. For a message queued at a PQ-node, $\omega_m$ is calculated by the following fixed-point iteration.

$$\omega_m^{n+1}(q_m) = B_m + q_m C_m + \sum_{\forall m_k \in hp(m_m)} I_k C_k \tag{6.7}$$

The last term in (6.7) represents the interference from the higher priority messages. In order to solve this iterative equation, initial value of $\omega_m^n$ can be taken as follows.

$$\omega_m^0(q_m) = B_m + q_m C_m \tag{6.8}$$

The iterations in (6.7) stop either when the queueing delays in the previous and current iterations are equal or when the response time exceeds the deadline. Since, CAN uses fixed priority non-preemptive scheduling, any message can be blocked by only one message in the set of lower priority messages. Hence, the message under analysis can only be blocked by either the periodic copy or the sporadic 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, sporadic and mixed messages in the set of lower priority messages with respect to $m_m$ and is given by the following equation.

$$B_m = \max_{\forall m_k \in lp(m_m)} (C_k) \tag{6.9}$$

A higher priority message $m_k$ contributes an extra delay, equal to $f_k$, to the worst-case queueing delay of $m_m$ if $m_k$ belongs to the FQ-node. $f_k$ represents

the delay after which the higher priority message $m_k$ belonging to the FQ-node becomes the oldest message in the queue and can take part in the priority-based arbitration [25]. The existing analysis for mixed messages in CAN [21] does not take this additional delay into account. $f_k$ is zero if $m_k$ belongs to a PQ-node. We will come back to the calculations for $f_k$ in Section 6.5.3.

In (6.7), $I_k$ is calculated differently for different values of $\xi_k$ (k is the index of any higher priority message) as shown below. The interference by a higher priority mixed message contains the contribution from both the duplicates.

$$I_k = \begin{cases} \left\lceil \frac{\omega_m^n(q_m)+J_k+f_k+\tau_{bit}}{T_k} \right\rceil, & \text{if } \xi_k = \text{P} \\ \left\lceil \frac{\omega_m^n(q_m)+J_k+f_k+\tau_{bit}}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{S} \\ \left\lceil \frac{\omega_m^n(q_m)+J_k+f_k+\tau_{bit}}{T_k} \right\rceil + & \\ \left\lceil \frac{\omega_m^n(q_m)+J_k+f_k+\tau_{bit}}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{M} \end{cases} \tag{6.10}$$

**Length of the maximum busy period.** The length of priority level-m maximum busy period, denoted by $t_m$, is given by the following equation. The effect of extra delay from the messages belonging to the FQ-nodes is also taken into account. $t_m$ can be calculated by the following iterative equation.

$$t_m^{n+1} = B_m + \sum_{\forall m_k \in hep(m_m)} I'_k C_k \tag{6.11}$$

$I'_k$ is given by the following relation. Note that the contribution of both the duplicates of a mixed message $m_k$ in the set $hep(m_m)$ is taken into account.

$$I'_k = \begin{cases} \left\lceil \frac{t_m^n+J_k+f_k}{T_k} \right\rceil, & \text{if } \xi_k = \text{P} \\ \left\lceil \frac{t_m^n+J_k+f_k}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{S} \\ \left\lceil \frac{t_m^n+J_k+f_k}{T_k} \right\rceil + \left\lceil \frac{t_m^n+J_k+f_k}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{M} \end{cases} \tag{6.12}$$

In order to solve the iterative equation (6.11), $C_m$ can be used as the initial value of $t_m^n$ as shown below.

$$t_m^0 = C_m \tag{6.13}$$

The right hand side of (6.11) is a monotonic non-decreasing function of $t_m$. Equation (6.11) 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,

$$U_m < 1 \tag{6.14}$$

where $U_m$ is calculated by the following equation:

$$U_m = \sum_{\forall m_k \in hep(m_m)} C_k I_k'' \tag{6.15}$$

where $I_k''$ is given by the following relation:

$$I_k'' = \begin{cases} \frac{1}{T_k}, & \text{if } \xi_k = \text{P} \\ \frac{1}{MUT_k}, & \text{if } \xi_k = \text{S} \\ \frac{1}{T_k} + \frac{1}{MUT_k}, & \text{if } \xi_k = \text{M} \end{cases} \tag{6.16}$$

In the above equation, the contribution by both the copies of all mixed messages belonging to the set $hep(m_m)$ is taken into account while calculating the bus utilization.

The number of instances of $m_m$, 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 for mixed messages).

$$Q_m = \begin{cases} \left\lceil \frac{t_m + J_m}{T_m} \right\rceil, & \text{if } \xi_m = \text{P} \\ \left\lceil \frac{t_m + J_m}{MUT_m} \right\rceil, & \text{if } \xi_m = \text{S} \end{cases} \tag{6.17}$$

The index of each message instance is identified by $q_m$ and its range is given as follows.

$$0 \leq q_m \leq (Q_m - 1) \tag{6.18}$$

### Case 1 (b): When $m_m$ Belongs to a FIFO-queued Node

Similar to the existing RTA for CAN with FIFO queues [25], the extended analysis is FIFO-symmetric. This means that all the messages belonging to FQ-node will have same upper bound for their worst-case response times. In order to derive the worst-case response time of a periodic or sporadic message belonging to the FQ-node, we consider the worst-case conditions. Hence, we assume that the message under analysis is the lowest priority message, i.e., $L_m$ in the group $M(m_m)$ with the largest transmission time $C_m^{MAX}$ (to maximize the interference from the messages in $M(m_m)$ as well as from the messages belonging to other nodes). The response time of a particular instance $q_m$ of a periodic or sporadic message $m_m$ that is queued at the FQ-node is given by the following equation.

$$R_m(q_m) = \begin{cases} J_m + \omega_m(q_m) - q_m T_m + C_m^{MAX}, & \text{if } \xi_m = \text{P} \\ J_m + \omega_m(q_m) - q_m MUT_m + C_m^{MAX}, & \text{if } \xi_m = \text{S} \end{cases} \tag{6.19}$$

In [25], message deadlines are assumed to be equal to or less than the corresponding periods. Hence, for any message $m_m$ belonging to $M(m_m)$ in the FQ-node, there could be only one instance of every other message queued ahead of $m_m$. In the existing analysis, the maximum amount of interference received by $m_m$ before it becomes the oldest message in the FIFO queue and ready to take part in the priority-based arbitration is bounded by ($C_m^{SUM} - C_m^{MIN}$). This interference bound may not be applicable in our case because we assume that the messages have arbitrary deadlines which means that they can be greater than the periods or minimum update times of the corresponding messages. Therefore, it is possible to have more than one instance of any higher priority message queued ahead of $m_m$ in the FIFO queue. This is the reason we select the transmission time of $m_m$ in FIFO-queued nodes to be equal to $C_m^{MAX}$ instead of $C_m^{MIN}$.

**Interference received by $m_m$ from the messages in $M(m_m)$.** Now, we derive an upper bound for the number of instances of each message in the group $M(m_m)$ that can be queued ahead of $m_m$. Consider a simple but intuitive example as shown in Figure 6.6. Let the message under analysis be $m_m$ (lowest priority message in $M(m_m)$). Also consider an arbitrary message $m_i$ belonging to the group $M(m_m)$. Assume both $m_i$ and $m_m$ are periodic and have same transmission times. We consider four different cases with respect to the

relationship between message periods as shown in Figure 6.6. In case (a), $T_i$ is smaller than $T_m$. In case (b), $T_i$ is equal to $T_m$. In case (c), $T_i$ is greater than $T_m$. In case (d), $T_i$ is smaller than $T_m$ and at the same time $T_m$ is an integer multiple of $T_i$. These cases essentially cover all the cases required to derive the upper bound on the maximum number of instances of $m_i$ queued ahead of any instance of $m_m$.

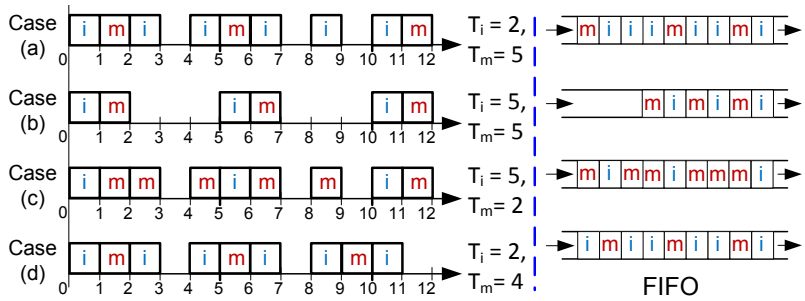


Figure 6.6: Demonstration of maximum interference on $m_m$ from the messages in the group $M(m_m)$.

The periods of $m_i$ and $m_m$ in each case are shown in Figure 6.6. The left hand side of Figure 6.6 shows the time line during which each instance of $m_i$ and $m_m$ is queued in the FIFO queue. Whereas, the right hand side of Figure 6.6 depicts the corresponding FIFO queue as if none of the messages was transmitted. The maximum number of instances of $m_i$ that are queued ahead of any instance of $m_m$ in the FIFOs are $3, 1, 1$ and $2$ in the case (a), (b), (c) and (d) respectively. Let $Q_i$ denotes the maximum number of instances of $m_i$ in the group $M(m_m)$ that can be queued ahead of any instance of $m_m$ in the FIFO queue. We can generalize $Q_i$ for all the cases as follows.

$$Q_i = \left\lceil \frac{T_m}{T_i} \right\rceil \tag{6.20}$$

Let us consider the effect of jitter of $m_i$, denoted by $J_i$, on the interference of $m_m$. Because of $J_i$, additional instances of $m_i$ can be queued ahead of $m_m$. Thus, taking the effect of jitter into account, (6.20) can be written as:

$$Q_i = \left\lceil \frac{T_m + J_i}{T_i} \right\rceil \tag{6.21}$$

Since, $m_i$ can be periodic, sporadic or mixed, we can generalize (6.21) as follows.

$$Q_i = \begin{cases} \left\lceil \frac{T_m+J_i}{T_i} \right\rceil, & \text{if } \xi_i = \text{P} \\ \left\lceil \frac{T_m+J_i}{MUT_i} \right\rceil, & \text{if } \xi_i = \text{S} \\ \left\lceil \frac{T_m+J_i}{T_i} \right\rceil + \left\lceil \frac{T_m+J_i}{MUT_i} \right\rceil, & \text{if } \xi_i = \text{M} \end{cases} \tag{6.22}$$

**Calculations for the worst-case queueing delay.** The worst-case queueing delay, $\omega_m$, in (6.19) can be calculated in a similar fashion as in (6.7) with the addition of extra delay shown in (6.22).

$$\begin{aligned} \omega_m^{n+1}(q_m) = B_{L_m} + & \sum_{\forall m_i \in M(m_m) \wedge i \neq m} Q_i C_i \\ + q_m C_m^{MAX} + & \sum_{\forall m_k \in hp(L_m) \wedge m_k \notin M(m_m)} I_k C_k \end{aligned} \tag{6.23}$$

Where $m_k$ is any message that has priority higher than the lowest priority message in the FQ-node in which $m_m$ is queued. Moreover, $m_k$ does not belong to the FQ-node in which $m_m$ is queued. $m_i$ is any message, other than $m_m$, in the group $M(m_m)$. $B_{L_m}$ is the blocking time of $L_m$ which refers to the maximum transmission time of a message in the set of messages with lower priority than $L_m$ that are sent by the other nodes. Since, the interference contributed to $m_m$ by higher priority messages from other nodes (both PQ and FQ) is independent of $m_m$ belonging to a PQ-node or FQ-node, $I_k$ can be calculated using (6.10). The initial value of $\omega_m^n$ to solve the iterative equation (6.23) can be selected as follows.

$$\omega_m^0 = B_{L_m} + \sum_{\forall m_i \in M(m_m) \wedge i \neq m} Q_i C_i + q_m C_m^{MAX} \tag{6.24}$$

**Length of the maximum busy period.** The length of priority level-m maximum busy period, denoted by $t_m$, can be calculated in a similar fashion as in (6.11) and by following the intuition from (6.23). The effect of extra delay from the messages belonging to the FQ-nodes is also taken into account. $t_m$ can be calculated by the following iterative equation.

$$t_m^{n+1} = B_{L_m} + \sum_{\forall m_i \in M(m_m) \wedge i \neq m} Q_i C_i + \sum_{\forall m_k \in hep(L_m) \wedge m_k \notin M(m_m)} I'_k C_k \quad (6.25)$$

The initial value for $t_m^n$ can be selected using (6.13). Since, the interference to $m_m$ by higher priority messages from other nodes (both PQ and FQ) is independent of $m_m$ belonging to a PQ-node or FQ-node, $I'_k$ can be calculated using (6.12). Similarly, the total number of instances of $m_m$ that becomes ready for transmission before the busy period ends can be calculated using (6.17). The worst-case response time of $m_m$ is the largest value of response time among all its instances as shown in (6.1).

### 6.5.2 Case 2: When $m_m$ is a Mixed Message

When the message under analysis is mixed, we treat it as two separate message streams, i.e., each mixed message is duplicated as the periodic and sporadic messages. The response times of both the duplicates are calculated separately. For simplicity, we denote the periodic and sporadic copies of a mixed message $m_m$ by $m_{m_P}$ and $m_{m_S}$ respectively. Let the worst-case response time of $m_{m_P}$ and $m_{m_S}$ be denoted by $R_{m_P}$ and $R_{m_S}$ respectively. The worst-case response time of $m_m$ is equal to the largest value between $R_{m_P}$ and $R_{m_S}$ as given by the following equation.

$$R_m = max(R_{m_P}, \ R_{m_S}) \quad (6.26)$$

#### Case 2 (a): When $m_m$ Belongs to a Priority-queued Node

For a priority-queued mixed message, the response times of each instance of $m_{m_P}$ and $m_{m_S}$ are calculated separately by adapting the existing analysis for mixed messages in CAN [21]. Let us denote the total number of instances of $m_{m_P}$ and $m_{m_S}$, occurring in the priority level-m maximum busy period, by $Q_{m_P}$ and $Q_{m_S}$ respectively. Assume that the index variable for message instances of $m_{m_P}$ and $m_{m_S}$ is denoted by $q_{m_P}$ and $q_{m_S}$ respectively. Their ranges are given by the following equations.

$$0 \leq q_{m_P} \leq (Q_{m_P} - 1) \quad (6.27)$$

$$0 \leq q_{m_S} \leq (Q_{m_S} - 1) \tag{6.28}$$

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

$$R_{m_P} = max(R_{m_P}(q_{m_P})) \tag{6.29}$$

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

$$R_{m_S} = max(R_{m_S}(q_{m_S})) \tag{6.30}$$

The worst-case response time of each instance of $m_{m_P}$ and $m_{m_S}$ can be derived by adapting the equations for the calculation of worst-case response time of periodic and sporadic messages respectively (derived in the first case) as given by the following two equations.

$$R_{m_P}(q_{m_P}) = J_m + \omega_{m_P}(q_{m_P}) - q_{m_P}T_m + C_m \tag{6.31}$$

$$R_{m_S}(q_{m_S}) = J_m + \omega_{m_S}(q_{m_S}) - q_{m_S}MUT_m + C_m \tag{6.32}$$

The queueing jitter, $J_m$, is the same (equal) in both the equations (6.31) and (6.32). The transmission time, $C_m$, is also the same in these equations and is calculated using (6.5) or (6.6) depending upon the type of CAN frame identifier. Although, both the duplicates of $m_m$ inherit same $J_m$ and $C_m$ from it, they experience different amount of worst-case queueing delays caused by other messages.

**Calculations for the worst-case queueing delay.** The worst-case queueing delay experienced by $m_{m_P}$ and $m_{m_S}$ is denoted by $\omega_{m_P}$ and $\omega_{m_S}$ in (6.31) and (6.32) respectively. $\omega_{m_P}$ and $\omega_{m_S}$ can be calculated by adapting the equation for the calculations of worst-case queueing delay in (6.7). However, in this equation we need to add the effect of self interference in a mixed message. By self interference we mean that the periodic copy of a mixed message can be interfered by the sporadic copy and vice versa. Since, both $m_{m_P}$ and $m_{m_S}$ have equal priorities, any number of instances of $m_{m_P}$ queued ahead of $m_{m_S}$ contribute an extra delay to the worst-case queueing delay experienced by $m_{m_S}$

and vice versa. We adapt the calculations for self interference in a mixed message that we derived in [21]. The worst-case queueing delay for $m_{m_P}$ and $m_{m_S}$ can be calculated using the following equations.

$$\omega_{m_P}^{n+1}(q_{m_P}) = B_m + q_{m_P}C_m + \sum_{\forall m_k \in hp(m_m)} I_{k_P}C_k + Q_{m_S}^P C_m \tag{6.33}$$

$$\omega_{m_S}^{n+1}(q_{m_S}) = B_m + q_{m_S}C_m + \sum_{\forall m_k \in hp(m_m)} I_{k_S}C_k + Q_{m_P}^S C_m \tag{6.34}$$

The effect of self interference can be seen in the last terms of (6.33) and (6.34). $Q_{m_S}^P$ denotes the total number of instances of $m_{m_S}$ that are queued ahead of $q_{m_P}^{th}$ instance of $m_{m_P}$. Similarly, $Q_{m_P}^S$ denotes the total number of instances of $m_{m_P}$ that are queued ahead of $q_{m_S}^{th}$ instance of $m_{m_S}$. The values of $Q_{m_S}^P$ and $Q_{m_P}^S$ are calculated as follows.

$$Q_{m_S}^P = \begin{cases} \left\lceil \frac{q_{m_P}T_m + J_m + \tau_{bit}}{MUT_m} \right\rceil, & \text{if } (q_{m_P} = 0) \;\&\&\; (J_m = 0) \\ \left\lceil \frac{q_{m_P}T_m + J_m}{MUT_m} \right\rceil, & \text{otherwise} \end{cases} \tag{6.35}$$

$$Q_{m_P}^S = \begin{cases} \left\lceil \frac{q_{m_S}MUT_m + J_m + \tau_{bit}}{T_m} \right\rceil, & \text{if } (q_{m_S} = 0) \;\&\&\; (J_m = 0) \\ \left\lceil \frac{q_{m_S}MUT_m + J_m}{T_m} \right\rceil, & \text{otherwise} \end{cases} \tag{6.36}$$

In order to solve the iterative equations (6.33) and (6.34), the initial values of $\omega_{m_P}^n(q_{m_P})$ and $\omega_{m_S}^n(q_{m_S})$ can be selected according to (6.8) in a similar fashion. $I_{k_P}$ and $I_{k_S}$ are calculated using to the following equations.

$$I_{k_P} = \begin{cases} \left\lceil \frac{\omega^n_{m_P}(q_{m_P}) + J_k + f_k + \tau_{bit}}{T_k} \right\rceil, & \text{if } \xi_k = \text{P} \\ \left\lceil \frac{\omega^n_{m_P}(q_{m_P}) + J_k + f_k + \tau_{bit}}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{S} \\ \left\lceil \frac{\omega^n_{m_P}(q_{m_P}) + J_k + f_k + \tau_{bit}}{T_k} \right\rceil + & \\ \left\lceil \frac{\omega^n_{m_P}(q_{m_P}) + J_k + f_k + \tau_{bit}}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{M} \end{cases} \tag{6.37}$$

$$I_{k_S} = \begin{cases} \left\lceil \frac{\omega^n_{m_S}(q_{m_S}) + J_k + f_k + \tau_{bit}}{T_k} \right\rceil, & \text{if } \xi_k = \text{P} \\ \left\lceil \frac{\omega^n_{m_S}(q_{m_S}) + J_k + f_k + \tau_{bit}}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{S} \\ \left\lceil \frac{\omega^n_{m_S}(q_{m_S}) + J_k + f_k + \tau_{bit}}{T_k} \right\rceil + & \\ \left\lceil \frac{\omega^n_{m_S}(q_{m_S}) + J_k + f_k + \tau_{bit}}{MUT_k} \right\rceil, & \text{if } \xi_k = \text{M} \end{cases} \tag{6.38}$$

The values of $I_{k_P}$ and $I_{k_S}$ in (6.37) and (6.38) differ from those calculated in [21] in a way that we consider an extra jitter, i.e., $f_k$ from every message that belongs to the FQ-node.

**Calculations for the length of the maximum busy period.** The length of priority level-m maximum busy period, denoted by $t_m$, can be calculated using (6.11) that is developed for periodic and sporadic messages in a PQ-node. This is because (6.11) 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 this equation. Therefore, there is no need to calculate $t_m$ for $m_{m_P}$ and $m_{m_S}$ separately. It should be calculated only once for a mixed message.

Although $t_m$ is the same for $m_{m_P}$ and $m_{m_S}$, the number of instances of both the messages that become ready for transmission just before the end of the maximum busy period, i.e., $Q_{m_P}$ and $Q_{m_S}$ respectively, may be different. The reason is that the calculations for $Q_{m_P}$ and $Q_{m_S}$ require $T_m$ and $MUT_m$ respectively and which may have different values. $Q_{m_P}$ and $Q_{m_S}$ can be calculated by adapting (6.17) that is derived for the calculation of the number

of instances of periodic and sporadic messages that become ready for transmission before the end of the busy period. $Q_{m_P}$ and $Q_{m_S}$ are given by the following equations.

$$Q_{m_P} = \left\lceil \frac{t_m + J_m}{T_m} \right\rceil \tag{6.39}$$

$$Q_{m_S} = \left\lceil \frac{t_m + J_m}{MUT_m} \right\rceil \tag{6.40}$$

**Case 2 (b): When $m_m$ Belongs to a FIFO-queued Node**

The worst-case response times of each instance of $m_{m_P}$ and $m_{m_S}$ queued at the FQ-node are calculated similar to the case of FIFO-queued messages that are periodic or sporadic.

$$R_{m_P}(q_{m_P}) = J_m + \omega_{m_P}(q_{m_P}) - q_{m_P}T_m + C_m^{MAX} \tag{6.41}$$

$$R_{m_S}(q_{m_S}) = J_m + \omega_{m_S}(q_{m_S}) - q_{m_S}MUT_m + C_m^{MAX} \tag{6.42}$$

**Calculations for the worst-case queueing delay.** The worst-case queueing delays for $m_{m_P}$ and $m_{m_S}$ are calculated by adapting the calculations in the equations (6.23), (6.33) and (6.34) as follows.

$$\omega_{m_P}^{n+1}(q_{m_P}) = B_{L_m} + \sum_{\forall m_i \in M(m_m) \wedge i \neq m} Q_i C_i + q_{m_P} C_m^{MAX} + \sum_{\forall m_k \in hp(L_m) \wedge m_k \notin M(m_m)} I_{k_P} C_k + Q_{m_S}^P C_m \tag{6.43}$$

$$\omega_{m_S}^{n+1}(q_{m_S}) = B_{L_m} + \sum_{\forall m_i \in M(m_m) \wedge i \neq m} Q_i C_i + q_{m_S} C_m^{MAX} + \sum_{\forall m_k \in hp(L_m) \wedge m_k \notin M(m_m)} I_{k_S} C_k + Q_{m_P}^S C_m \tag{6.44}$$

Since, the interference caused by higher priority messages from other PQ- and FQ-nodes is independent of the mixed message $m_m$ belonging to a PQ-node or FQ-node, $I_{k_P}$ and $I_{k_S}$ can be calculated using (6.37) and (6.38). The initial values of $\omega_{m_P}$ and $\omega_{m_S}$ can be selected according to (6.24) while considering the respective index of each instance of $m_{m_P}$ and $m_{m_S}$. The value of $Q_i$ is calculated using (6.22) similar to the case of FIFO queued periodic or sporadic messages. The values of $Q^P_{m_S}$ and $Q^S_{m_P}$ are calculated using (6.35) and (6.36) that are derived for mixed message in a PQ-node. $Q^P_{m_S}$ denotes the total number of instances of $m_S$ that are queued ahead of $q^{th}_{m_P}$ instance of $m_P$. Therefore, we consider only queueing jitter in (6.35) and do not take into account any additional delay that may occur after queueing of $m_{m_P}$ such as $f_m$. Similar arguments hold for $Q^S_{m_P}$.

**Calculations for the length of the maximum busy period.** The length of priority level-m maximum busy period, denoted by $t_m$, can be calculated by using (6.25) that is developed for periodic and sporadic messages in a FQ-node. This is because (6.25) 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 the queueing delay from the duplicate is also covered in (6.25). Therefore, there is no need to calculate $t_m$ for $m_{m_P}$ and $m_{m_S}$ separately. It should be calculated only once for a mixed message.

Although the length of the busy period is the same, the number of instances of $m_{m_P}$ and $m_{m_S}$ that become ready for transmission just before the end of the maximum busy period, i.e., $Q_{m_P}$ and $Q_{m_S}$ respectively, may be different. $Q_{m_P}$ and $Q_{m_S}$ can be calculated by following the same reasoning and using the equations that we derived for a mixed message in the PQ-node in (6.39) and (6.40) respectively.

### 6.5.3 Algorithm for the Calculations of Maximum Buffering Time in FIFO Queues

The algorithm for the calculations of maximum buffering time in FIFO queues is adapted from [25] to support mixed messages in CAN with FIFO queues. The buffering time for any priority-queued message is equal to zero. It should be noted that the calculations for the response times in equations (6.2), (6.19), (6.31), (6.32), (6.41) and (6.42) are dependent upon the corresponding iterative calculations for the queueing delays in (6.7), (6.23), (6.33), (6.34), (6.43) and (6.44) respectively. Whereas, the calculations for queueing delay depends upon the maximum buffering time. Therefore, the response times and maximum

buffering times should be calculated iteratively and simultaneously as shown in Algorithm 1.

## 6.6 Case Study and Evaluation

In this section, we conduct an automotive-application case study. Basically, we adapt the case study of the experimental vehicle that is analyzed for only periodic messages in [32]. We implemented[4] the extended analysis in a freely-available tool MPS-CAN Analyzer [27]. Using this tool, we compare and evaluate the response times of periodic, sporadic and mixed messages in the experimental vehicle using the extended analysis for mixed messages in CAN with FIFO queues and the existing analysis for CAN with priority queues.

### 6.6.1 Experimental Setup

There are six ECUs in the experimental vehicle that are connected to a single CAN network. The selected speed for CAN is 500 Kbit/s. There are 81 CAN messages in the system; out of which 27 are periodic, 27 are sporadic, while the remaining 27 are mixed. All the attributes of these messages are shown in the table depicted in Figure 6.7. The attributes of each message are identified as follows. The priority, transmission type, number of data bytes in the message, transmission period, and minimum update time are represented by $P_m$, $\xi_m$, $s_m$, $T_m$, and $MUT_m$ respectively. We assume, the smaller the value of the $P_m$ parameter of a message, the higher its priority. Accordingly, the message with priority 1 is the highest priority message, whereas the message with priority 81 is the lowest priority message in the system. All timing parameters are in microseconds. We perform two sets of experiments. In the first set, all ECUs in the system implement priority queues. In the second set of experiments, all ECUs implement FIFO queues. In both sets of experiments, each ECU has 32 buffers in the transmit queue.

### 6.6.2 Comparison of Various Response-time Analyses

In the first set of experiments, the response times of all messages are calculated using the existing response-time analysis for mixed, periodic and sporadic messages in CAN with priority queues [21]. The calculated response times are denoted by $R_m[Prio]$ in the table in Figure 6.7. On the other hand, in the second

[4]The discussion about the implementation in the tool is not in the scope of this paper.

**Algorithm 1** Algorithm for the calculations of maximum buffering times and message response times simultaneously.

**begin**
**for all** Messages in the system **do**
    $f_m \leftarrow 0$ ▷ Initialize buffering time for all messages.
**end for**
$Repeat \leftarrow TRUE$
**while** $Repeat = TRUE$ **do**
    REPEAT ← FALSE
    **for** Every message $m_m$ in the system **do**
        **if** $m_m \in$ ECU with FIFO queue **then**
            **if** Message type of $m_m ==$ PERIODIC or SPORADIC **then**
                CALCULATE $R_m$ USING EQUATION (6.19)
            **else if** Message type of $m_m ==$ MIXED **then**
                CALCULATE $R_m$ USING EQUATIONS (6.41) AND (6.42)
            **end if**
            **if** $R_m > D_m$ **then**
                $m_m$ IS UNSCHEDULABLE
            **end if**
            **if** $f_m < \omega_m$ **then**
                $f_m \leftarrow \omega_m$
                REPEAT ← TRUE
            **end if**
        **else if** $m_m \in$ ECU with priority queue **then**
            $f_m \leftarrow 0$ ▷ buffering time for a priority queued message is always zero.
            **if** Message type of $m_m ==$ PERIODIC or SPORADIC **then**
                CALCULATE $R_m$ USING EQUATION (6.2)
            **else if** Message type of $m_m ==$ MIXED **then**
                CALCULATE $R_m$ USING EQUATIONS (6.31) AND (6.32)
            **end if**
            **if** $R_m > D_m$ **then**
                $m_m$ IS UNSCHEDULABLE
            **end if**
        **end if**
    **end for**
**end while**
**end**

| $P_m$ | $\xi_{sm}$ | $s_m$ | $T_m$ (us) | $MUT_m$ (us) | $R_m$[Pio] (us) | $R_m$[FIFO] (us) |
|---|---|---|---|---|---|---|
| 1 | P | 8 | 12500 | 0 | 540 | 32440 |
| 2 | S | 8 | 0 | 12500 | 810 | 39210 |
| 3 | M | 8 | 12500 | 12500 | 1080 | 34870 |
| 4 | S | 8 | 0 | 12500 | 1620 | 32980 |
| 5 | S | 8 | 0 | 50000 | 1890 | 35410 |
| 6 | M | 8 | 50000 | 50000 | 2160 | 35450 |
| 7 | S | 8 | 0 | 100000 | 2700 | 42860 |
| 8 | S | 8 | 0 | 20000 | 2970 | 35410 |
| 9 | M | 8 | 50000 | 50000 | 3240 | 33000 |
| 10 | S | 8 | 0 | 125000 | 3780 | 35330 |
| 11 | S | 8 | 0 | 25000 | 4050 | 42860 |
| 12 | S | 3 | 0 | 10000000 | 4220 | 43140 |
| 13 | M | 8 | 100000 | 100000 | 4490 | 42980 |
| 14 | P | 8 | 100000 | 0 | 5030 | 40020 |
| 15 | M | 8 | 100000 | 100000 | 5300 | 42980 |
| 16 | M | 8 | 100000 | 100000 | 5840 | 42980 |
| 17 | S | 8 | 0 | 100000 | 6380 | 33250 |
| 18 | P | 8 | 1000000 | 0 | 6650 | 33250 |
| 19 | S | 8 | 0 | 1000000 | 6920 | 40020 |
| 20 | P | 8 | 1000000 | 0 | 7190 | 35600 |
| 21 | P | 8 | 1000000 | 0 | 7460 | 33250 |
| 22 | M | 8 | 500000 | 500000 | 7730 | 35720 |
| 23 | P | 8 | 500000 | 0 | 8270 | 35600 |
| 24 | S | 8 | 0 | 500000 | 8540 | 43140 |
| 25 | P | 8 | 500000 | 0 | 8810 | 40020 |
| 26 | P | 8 | 100000 | 0 | 9080 | 35680 |
| 27 | S | 8 | 0 | 100000 | 9350 | 43140 |
| 28 | P | 8 | 100000 | 0 | 9620 | 35600 |
| 29 | S | 8 | 0 | 1000000 | 9890 | 43140 |
| 30 | M | 8 | 1000000 | 1000000 | 10160 | 33270 |
| 31 | S | 8 | 0 | 1000000 | 10700 | 33250 |
| 32 | M | 8 | 20000 | 20000 | 10970 | 35680 |
| 33 | S | 8 | 0 | 50000 | 11510 | 35600 |
| 34 | M | 8 | 500000 | 500000 | 11780 | 33270 |
| 35 | P | 8 | 20000 | 0 | 12320 | 33250 |
| 36 | P | 8 | 500000 | 0 | 12590 | 40020 |
| 37 | P | 8 | 20000 | 0 | 14210 | 33250 |
| 38 | S | 8 | 0 | 200000 | 14480 | 42860 |
| 39 | P | 8 | 20000 | 0 | 14750 | 43140 |
| 40 | P | 8 | 200000 | 0 | 15020 | 35600 |
| 41 | P | 8 | 1000000 | 0 | 15290 | 43140 |
| 42 | P | 8 | 100000 | 0 | 15560 | 35600 |
| 43 | S | 8 | 0 | 100000 | 15830 | 40020 |
| 44 | P | 8 | 100000 | 0 | 16100 | 42860 |
| 45 | S | 8 | 0 | 50000 | 16370 | 33250 |
| 46 | P | 8 | 50000 | 0 | 16640 | 40020 |
| 47 | S | 8 | 0 | 50000 | 16910 | 35600 |
| 48 | M | 8 | 50000 | 50000 | 17180 | 40160 |
| 49 | S | 8 | 0 | 1000000 | 17720 | 35600 |
| 50 | P | 8 | 1000000 | 0 | 17990 | 43140 |
| 51 | S | 8 | 0 | 1000000 | 18260 | 40020 |
| 52 | P | 8 | 1000000 | 0 | 18530 | 42860 |
| 53 | M | 8 | 128000 | 128000 | 18800 | 43260 |
| 54 | S | 8 | 0 | 128000 | 19340 | 35680 |
| 55 | P | 8 | 128000 | 0 | 19610 | 35600 |
| 56 | M | 8 | 1000000 | 1000000 | 19880 | 40160 |
| 57 | S | 8 | 0 | 250000 | 22040 | 40020 |
| 58 | M | 3 | 250000 | 250000 | 22210 | 43160 |
| 59 | M | 8 | 500000 | 500000 | 22650 | 40160 |
| 60 | M | 8 | 500000 | 500000 | 23190 | 35680 |
| 61 | M | 7 | 500000 | 500000 | 23710 | 33270 |
| 62 | M | 8 | 500000 | 500000 | 24230 | 35720 |
| 63 | S | 2 | 0 | 500000 | 24650 | 35720 |
| 64 | M | 8 | 1000000 | 1000000 | 24920 | 35720 |
| 65 | P | 8 | 1000000 | 0 | 27080 | 35680 |
| 66 | M | 8 | 1000000 | 1000000 | 27350 | 35680 |
| 67 | P | 8 | 1000000 | 0 | 27890 | 35680 |
| 68 | P | 8 | 1000000 | 0 | 28160 | 43140 |
| 69 | P | 6 | 1000000 | 0 | 28390 | 42860 |
| 70 | S | 8 | 0 | 2000000 | 28660 | 33250 |
| 71 | S | 8 | 0 | 2000000 | 28930 | 42860 |
| 72 | P | 8 | 2000000 | 0 | 29200 | 43140 |
| 73 | M | 8 | 2000000 | 2000000 | 29470 | 43260 |
| 74 | M | 8 | 2000000 | 2000000 | 30010 | 40160 |
| 75 | S | 8 | 0 | 2000000 | 30550 | 35680 |
| 76 | P | 8 | 2000000 | 0 | 30820 | 42860 |
| 77 | M | 8 | 2000000 | 2000000 | 31090 | 35680 |
| 78 | M | 2 | 2000000 | 2000000 | 31390 | 42860 |
| 79 | M | 1 | 50000 | 50000 | 31670 | 40040 |
| 80 | M | 2 | 1000000 | 1000000 | 31950 | 42860 |
| 81 | M | 2 | 2000000 | 2000000 | 32250 | 43140 |

Figure 6.7: Attributes and calculated response times of the periodic, sporadic and mixed messages in the experimental vehicle.

set of experiments, the response times of all messages are calculated using the extended analysis presented in the previous section. The calculated response times in this case are denoted by $R_m[FIFO]$ in the table in Figure 6.7. The maximum network bandwidth utilization calculated in both cases is equal to 33.776970%.

The response times of all messages in these two cases are shown by the bar graphs in Figure 6.8. The first bar (solid black bar) in each set of the two bars represents the response time of a message in the system where all ECU's implement priority queues. Whereas, the second bar (pattern bar) in each set of the two bars represents the response time of a message in the system where

all ECUs implement FIFO queues.

The response-time graphs show that the message response times are the best (smallest) in the case when all the ECUs use priority-based queueing policy. On the other hand, the response times of the messages are the worst (largest) in the system where the ECUs implement FIFO queues. In fact, the response times are significantly large in the case of FIFO queues. This is because of the priority inversion in FIFO queues (discussed in Section 6.3.3). Moreover, the worst-case buffering time in the FIFO queues can be significantly large that adds to the worst-case response times of the messages.

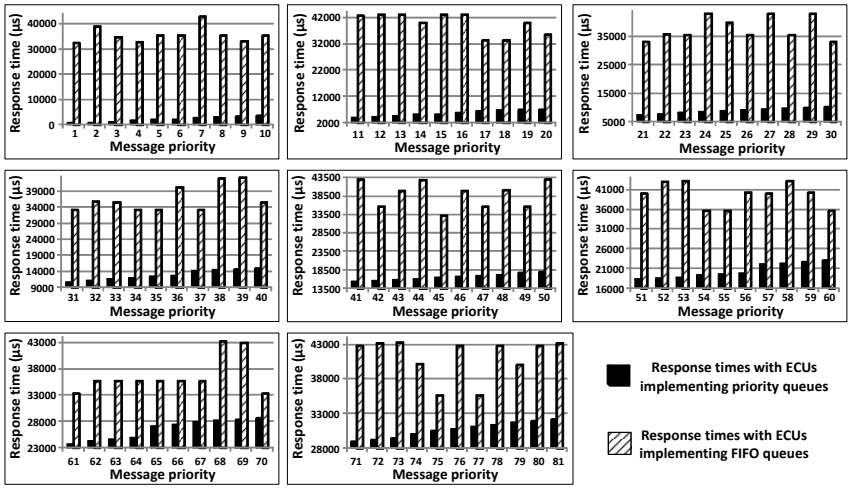


Figure 6.8: Comparison of message response-times with respect to different types of queueing policies in the ECUs.

### 6.6.3 Discussion and Recommendations

In order to get short response times of CAN messages, those ECUs should be selected that implement priority-based queueing policy. Although FIFO policy is simple and easy to implement, configure, and use as compared to the priority queueing policy, the messages can have very large worst-case response times in the case of ECUs implementing FIFO queues as shown in Figure 6.8. The ECUs which implement priority-based queueing policy should be preferred over the ECUs which implement FIFO queues especially in the sys-

tems that have high network utilization and short stimulus-to-response requirements. Moreover, it is important to use the right RTA that correctly matches the queueing policies in the ECUs; and transmission type of messages used in the higher-level protocols. If these constraints are not rightly considered in the RTA, the calculated response times can be optimistic.

## 6.7 Summary and Conclusion

The existing worst-case response-time analysis for messages in Controller Area Network (CAN) with priority- and FIFO-queued nodes does not support the analysis of mixed messages. A mixed message can be queued both periodically and sporadically, i.e., it may not have a periodic activation pattern. Mixed messages are implemented by several high-level protocols based on CAN that are used in the automotive industry. We identified three different implementations of mixed messages in higher-level protocols for CAN. For some implementations, the existing analysis still provides safe upper bounds for worst-case response times. Whereas for the others, the existing analysis calculates optimistic worst-case response times.

We extended the existing analysis for CAN with FIFO queues to provide safe upper bounds on the worst-case response times of mixed messages. The extended analysis is generally applicable to any higher-level protocol for CAN that supports periodic, sporadic, and mixed transmission of messages in a system comprising of priority- and FIFO-queued nodes. We conducted a case study and performed comparative evaluation of the extended analysis with the existing analysis for mixed, periodic and sporadic messages in CAN with priority queues.

The FIFO queues are already used in practical CAN controllers. Although, they are easy to implement and use, they can result in higher response times of messages. Therefore, the CAN controllers which implement priority queues should be preferred over the CAN controllers that implement FIFO queues. Moreover, it is important to use the response-time analysis that correctly matches the queueing policies in the ECUs; and transmission types of messages used in the higher-level protocols. If these constraints are not rightly considered in the response-time analysis, the calculated response times can be optimistic.

# Acknowledgement

This work is supported by the Swedish Knowledge Foundation (KKS) within the projects FEMMVA and EEMDEF, the Swedish Research Council (VR) within projects SynthSoft and TiPCES, and the Strategic Research Foundation (SSF) with the centre PROGRESS. The authors would like to thank the industrial partners Arcticus Systems AB, BAE Systems Hägglunds and Volvo Construction Equipment (VCE), Sweden.


# Appendix A

| Notation | Explanation |
|---|---|
| $m_n$ | Any message $_n$ |
| $ID_n$ | Unique identifier of $m_n$ |
| $P_n$ | Priority of $m_n$ |
| $\xi_n$ | Transmission type of $m_n$. It specifies whether $m_n$ is periodic ($P$), sporadic ($S$) or mixed ($M$) |
| $C_n$ | Worst-case transmission time of $m_n$ |
| $J_n$ | Queueing jitter of $m_n$ |
| $s_n$ | Size of data payload in $m_n$ |
| $T_n$ | Transmission period of $m_n$ |
| $MUT_n$ | Minimum Update Time of $m_n$. It is the minimum time that should elapse between the transmission of any two sporadic messages |
| $B_n$ | Blocking time of $m_n$ |
| $R_n$ | Worst-case response time of $m_n$ |
| $D_n$ | Deadline of $m_n$ |
| $hp(m_n)$ | Set of higher priority messages than $m_n$ |
| $lp(m_n)$ | Set of lower priority messages than $m_n$ |
| $hep(m_n)$ | Set of higher and equal priority messages than $m_n$ |
| $lep(m_n)$ | Set of lower and equal priority messages than $m_n$ |
| $\omega_n$ | Queueing delay for $m_n$ |
| $f_n$ | Maximum buffering time for $m_n$ |
| $\tau_{bit}$ | Time required to transmit a single bit of data over CAN |
| $t_n$ | Length of the priority level-n busy period |
| $q_n$ | Index variable to denote multiple instances of $m_n$ |
| $U_n$ | Bus utilization for priority level-n |

| | |
|---|---|
| $Q_n$ | Total Number of instances of $m_n$ that are queued in priority level-n busy period |
| $M(m_n)$ | The set of FIFO-queued messages in the sender ECU of $m_n$ |
| $L_n$ | The lowest priority message in the set $M(m_n)$ |
| $B_{L_n}$ | Blocking time due to $L_n$ |
| $C_n^{MAX}$ | Maximum transmission time of a message in the set $M(m_n)$ |
| $C_n^{MIN}$ | Minimum transmission time of a message in the set $M(m_n)$ |
| $m_{n_P}$ | Periodic part of a mixed message $m_n$ |
| $m_{n_S}$ | Sporadic part of a mixed message $m_n$ |
| $R_{n_P}$ | Response time of $m_{n_P}$ |
| $R_{n_S}$ | Response time of $m_{n_S}$ |

Table 6.1: Notations and terminology.

# Bibliography


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

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

[3] Robert Bosch GmbH. CAN with Flexible Data-Rate (CAN FD), White Paper, Ver. 1.1. 2011.

[4] Automotive networks. CAN in Automation (CiA), 2011. http://www.can-cia.org/index.php?id=416, accessed on Feb. 05, 2014.

[5] Peter Thorngren. Keynote Talk: Experiences from EAST-ADL Use, EAST-ADL Open Workshop, Gothenberg, Oct., 2013.

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

[7] CANopen Application Layer and Communication Profile. CiA Draft Standard 301. Ver. 4.02. Feb., 2002. http://www.can-cia.org/index.php?id=440, accessed on Feb. 05, 2014.

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

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

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

[11] 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.

[12] 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.

[13] M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer Journal,*, 29(5):390–395, 1986.

[14] Mikael Nolin, Jukka Mäki-Turja, and Kaj Hänninen. Achieving industrial strength timing predictions of embedded system behavior. In *International Conference on Embedded Systems and Applications (ESA), 2008*, pages 173–178, 2008.

[15] S. Mubeen, J. Mäki-Turja and M. Sjödin. Response-time analysis of mixed messages in Controller Area Network with priority- and FIFO-queued nodes. In *9th IEEE International Workshop on Factory Communication Systems (WFCS), May, 2012*, May 2012.

[16] 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.

[17] Volcano Network Architect. Mentor Graphics, http://www.mentor.com/products/vnd/communication-management/vna, accessed on Feb. 05, 2014.

[18] 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.

[19] Rubus-ICE: Integrated component Development Environment, http://www.arcticus-systems.com, accessed on feb. 05, 2014.

[20] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Support for end-to-end response-time and delay analysis in the industrial tool suite: Issues, experiences and a case study. *Computer Science and Information Systems, ISSN: 1361-1384,*, 10(1), 2013.

[21] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Extending schedulability analysis of Controller Area Network (CAN) for mixed (periodic/sporadic) messages. In *16th IEEE Conference on Emerging Technologies and Factory Automation (ETFA), Sep., 2011*, Sep. 2011.

[22] Rob Davis, Steffen Kollmann, Victor Pollex, and Frank Slomka. Schedulability analysis for controller area network (CAN) with FIFO queues priority queues and gateways. *Real-Time Systems*, 49(1):73–116, 2013.

[23] D.A. Khan, R.J. Bril, and N. Navet. Integrating hardware limitations in CAN schedulability analysis. In *8th IEEE International Workshop on Factory Communication Systems (WFCS), May, 2010*, pages 207 –210, May 2010.

[24] Marco Di Natale, Haibo Zeng, Paolo Giusto, Arkadeb Ghosal. *Understanding and Using the Controller Area Network Communication Protocol*. Springer, 2012.

[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, July, 2011*, July 2011.

[26] R. Davis and N. Navet. Controller area network (CAN) schedulability analysis for messages with arbitrary deadlines in FIFO and work-conserving queues. In *9th IEEE International Workshop on Factory Communication Systems (WFCS), May, 2012*, pages 33 –42, May 2012.

[27] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Many-in-one Response-Time Analyzer for Controller Area Network. In *4th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, July 2013.

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

[29] Requirements on Communication, Rel. 4.1, Rev. 3, Ver. 3.3.1, March, 2014. www.autosar.org/download/R4.1/AUTOSAR_SRS_COM.pdf, accessed on May 05, 2014.

[30] AUTOSAR Techincal Overview, Release 4.1, Rev. 2, Ver. 1.1.0., The AUTOSAR Consortium, Oct., 2013, 2013. http://autosar.org.

[31] Ian Broster. *Flexibility in Dependable Real-time Communication*. PhD thesis, University of York, August 2003.

[32] Marco Di Natale and Haibo Zeng. Practical issues with the timing analysis of the Controller Area Network. In *18th IEEE Conference on Emerging Technologies and Factory Automation (ETFA), Sep., 2013*, Sep. 2013.

# Chapter 7

# Paper D: Extending offset-based response-time analysis for mixed messages in Controller Area Network

Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin
In the 18th IEEE conference on Emerging Technologies and Factory Automation (ETFA), September, 2013.
**IEEE Industrial Electronics Society Scholarship Award.**

**Abstract**

The existing offset-based response-time analysis for mixed messages in Controller Area Network (CAN) assumes the jitter and deadline of a message to be smaller or equal to the transmission period. However, practical systems may contain messages whose release jitter and deadlines can be greater than their periods, e.g., in the gateway nodes. We extend the existing response-time analysis for mixed messages in CAN that are scheduled with offsets and have arbitrary jitter and deadlines. Mixed messages are implemented by several higher-level protocols for CAN that are used in the automotive industry. The extended analysis is applicable to any higher-level protocol for CAN that uses periodic, sporadic and mixed transmission modes.

## 7.1 Introduction

The Controller Area Network (CAN) [1] is a multi-master, event-triggered, serial communication bus protocol supporting bus speeds of up to 1 mega bits per second. It is a widely used protocol in the automotive domain. It has been standardized as ISO 11898-1 [2]. According to CAN in Automation (CiA) [3], the estimated number of CAN enabled controllers sold in 2011 are about 850 million. CAN also finds its applications in other domains, e.g., industrial control, medical equipments, maritime electronics, and production machinery. There are several higher-level protocols for CAN that are developed for many industrial applications such as CAN Application Layer (CAL), CANopen, J1939, Hägglunds Controller Area Network (HCAN), CAN for Military Land Systems domain (MilCAN).

In order to provide evidence that each action 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, *a priori* analysis techniques such as schedulability analysis have been developed by the research community. Response-Time Analysis (RTA) [4, 5, 6, 7] 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 network respectively. 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) [8].

### 7.1.1 Motivation and Related Work

Tindell et al. [9] developed the schedulability analysis for CAN. This analysis has been implemented in several tools that are used in the automotive industry [10, 11, 12, 13]. Davis et al. [14] refuted, revisited and revised the analysis by Tindell et al. In [15], Davis et al. extended the analysis in [9, 14] which is applicable to the CAN network where some nodes implement priority queues and some implement FIFO queues. All these analyses assume that the messages are queued for transmission periodically or sporadically. They do not support mixed messages in CAN, i.e., the messages that are simultaneously time (periodic) and event (sporadic) triggered. Mubeen et al. [16] extended the existing analysis to support mixed messages in CAN where nodes implement

priority-based queues. Mubeen et al. [17] further extended their analysis to support mixed messages in the network where some nodes implement priority queues while others implement FIFO queues.

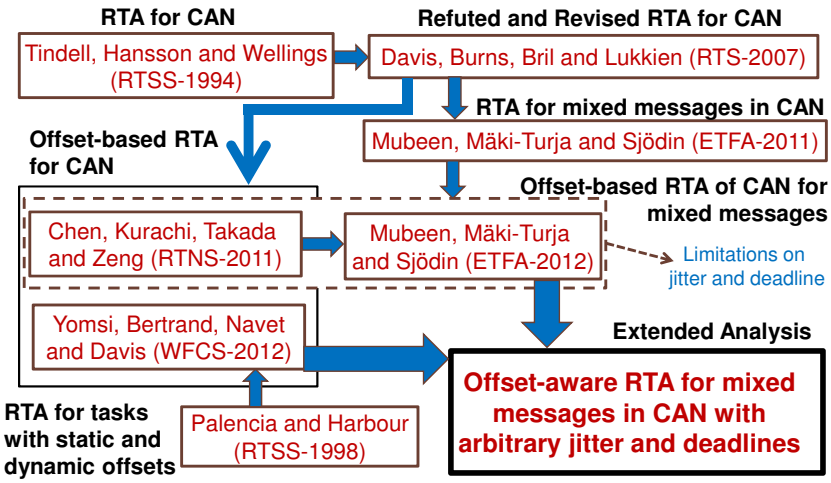


Figure 7.1: Relation between the existing and extended Response Time Analysis (RTA)

But, none of the analysis discussed above supports messages that are scheduled with offsets, i.e., using externally imposed delays between the times when the messages can be queued. In order to avoid deadlines violations due to high transient loads, current automotive embedded systems are often scheduled with offsets [18]. Furthermore, the worst-case response times of messages (especially with lower priority) in CAN increase with the increase in the network load. However, the worst-case response-times of lower priority messages in CAN can be reduced if the messages are scheduled with offsets [19, 20, 21]. A method for the assignment of offsets to improve the overall bandwidth utilization is proposed in [20, 21]. The worst-case response-time analysis for CAN messages with offsets has been developed by several researchers [22, 23, 19, 24, 18].

None of these analyses support mixed messages that are scheduled with offsets. In [25], we extended the existing offset-based analysis [22] to support worst-case response-time calculations for mixed messages in CAN. However, this analysis is restricted due to limitations regarding message jitter and dead-

lines. The source of these limitations comes from the base analysis [22]. In this paper, we remove these limitations. Basically, we extend the analysis for mixed messages [16] by building it upon the analysis for CAN messages with offsets [18]. Figure 7.1 depicts the relation between the existing and extended analyses.

### 7.1.2 Paper Contribution

We extend the response-time analysis of CAN for mixed messages that are scheduled with offsets. The existing analysis for mixed messages with offsets [25] places restrictions on message deadline and jitter, i.e., each of them should be less than or equal to message period. Release jitter of a message can be higher than its transmission period in practical systems. For example, consider a message is exchanged between two networks (or two segments of a network) via a gateway node. At the gateway node, the message inherits its response time on the incoming network as part of its release jitter on the outgoing network. This release jitter can be higher than the period of the message [18]. The existing offset-based analysis does not support mixed messages whose jitter and deadlines are higher than their transmission periods.

In this paper, we lift these restrictions by assuming deadline and jitter to be arbitrary, i.e., each one of them can be higher than message period. Intuitively, there can be several instances of the same message that are queued for transmission. Hence, our extended analysis considers the response times of all these instances while calculating the worst-case response time. Mixed messages are implemented by several higher-level protocols used in the industry. The analysis is applicable to any higher-level protocol for CAN that uses periodic, sporadic and mixed transmission of messages that are scheduled with offsets. We also show the applicability of the extended analysis by conducting the automotive-application case study.

### 7.1.3 Paper Layout

The remainder of the paper is organized as follows. In Section 7.2, we discuss mixed transmission patterns supported by higher-level protocols for CAN. Section 7.3 describes the scheduling model. Section 7.4 presents the extended analysis. In Section 7.5, we present a case study. Section 7.6 concludes the paper and discusses the future work.

## 7.2 Mixed Transmission Patterns Supported by Higher-level Protocols

When CAN is employed for network communication in a distributed real-time system, each node (processor) or Electronic Control Unit (ECU) is equipped with a CAN interface that connects the node to the bus [26]. Application tasks in each node, that require remote transmission, are assumed to queue messages for transmission over the CAN network. The messages are transmitted according to the protocol specification of the CAN protocol. Traditionally, it is assumed 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 some higher-level protocols and commercial extensions of CAN in which the task that queues the messages can be invoked periodically as well as sporadically. If a message can be queued for transmission periodically as well as at the arrival of a sporadic event then the transmission type of the message is said to be mixed. In other words, a mixed message is simultaneously time (periodic) and event triggered (sporadic). We identified three types of implementations of mixed messages used in the industry.

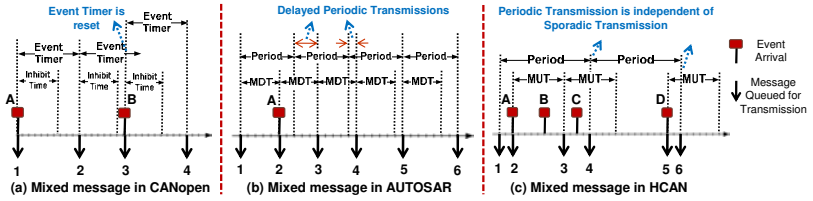


Figure 7.2: Mixed transmission pattern in higher-level protocols for CAN

**Consistent terminology.** To stay consistent, we use the terms message and frame interchangeably because we only consider messages that will fit into one frame (maximum 8 bytes). For the purpose of using simple notation, we call a CAN message as periodic, sporadic or mixed if it is queued by an application task that is invoked periodically, sporadically or both (periodically and sporadically) respectively. If a message is queued for transmission at periodic intervals, we use the term "Period" to refer to its periodicity. A sporadic message is queued for transmission as soon as an event occurs that changes the value of one or more signals contained in the message provided the Minimum Update Time ($MUT$) between queueing of two successive sporadic messages has elapsed. Hence, the transmission of a sporadic frame is constrained by

the $MUT$. We overload the term "$MUT$" to refer to the "Inhibit Time" in CANopen protocol [27] and the "Minimum Delay Time (MDT)" in AUTOSAR communication [28].

### 7.2.1 Method 1: Implementation in CANopen

The CANopen protocol [27] supports mixed transmission that corresponds to the Asynchronous Transmission Mode coupled with the Event Timer. The Event Timer is used to transmit an asynchronous message cyclically. A mixed message can be queued for transmission at the arrival of an event provided the 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 the Event Timer. The Event Timer is reset every time the message is queued. Once a mixed message is queued, any additional queueing of it will not take place during the Inhibit Time [27].

The transmission pattern of a mixed message in CANopen is illustrated in Figure 7.2(a). The down-pointing arrows symbolize the queueing of messages while the upward lines (labeled with alphabetic characters) represent arrival of the events. Message 1 is queued as soon as the event $A$ arrives. Both the Event Timer and Inhibit Time are reset. As soon as the Event Timer expires, message 2 is queued due to periodicity and both the Event Timer and Inhibit Time are reset again. When the event $B$ arrives, message 3 is immediately queued because the Inhibit Time has already expired. Note that the Event Timer is also reset at the same time when message 3 is queued as shown in Figure 7.2(a). Message 4 is queued because of the expiry of the Event Timer. There exists a dependency relationship between the Inhibit Time and the Event Timer, i.e., the Event Timer is reset not only with every periodic transmission but also with every sporadic transmission.

### 7.2.2 Method 2: Implementation in AUTOSAR

AUTOSAR (AUTomotive Open System ARchitecture) [29] can be viewed as a higher-level protocol if it uses CAN for network communication. Mixed transmission mode in AUTOSAR is widely used in practice. In AUTOSAR, a mixed message can be queued for transmission repeatedly with a period equal to the mixed transmission mode time period. The mixed message can also be queued at the arrival of an event provided the Minimum Delay Time ($MDT$) has been expired. However, each transmission of the mixed message, regardless of be-

ing periodic or sporadic, is limited by the $MDT$. This means that both periodic and sporadic transmissions are delayed until the $MDT$ expires. The transmission pattern of a mixed message implemented by AUTOSAR is illustrated in Figure 7.2(b). Message 1 is queued (the $MDT$ is started) because of partly periodic nature of the mixed message. When the event $A$ arrives, message 2 is queued immediately because the $MDT$ has already expired. The next periodic transmission is scheduled 2 time units after the transmission of message 2. However, the next two periodic transmissions corresponding to messages 3 and 4 are delayed because the $MDT$ is not expired. This is indicated by the comment "Delayed Periodic Transmissions" in Figure 7.2(b). The periodic transmissions corresponding to messages 5 and 6 occur at the scheduled times because the $MDT$ is already expired in both cases.

### 7.2.3 Method 3: Implementation in HCAN

A mixed message in HCAN protocol [30] contains signals out of which some are periodic and some are sporadic. 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 the $MUT$ between the queueing of two successive sporadic instances of the mixed message has elapsed. Hence, the transmission of the mixed message due to arrival of events is constrained by the $MUT$. The transmission pattern of the mixed message is illustrated in Figure 7.2(c). Message 1 is queued because of periodicity. As soon as event $A$ arrives, message 2 is queued. When event $B$ arrives it is not queued immediately because the $MUT$ is not expired yet. As soon as the $MUT$ expires, message 3 is queued. Message 3 contains the signal changes that correspond to event $B$. Similarly, a message is not immediately queued when the event $C$ arrives because the $MUT$ is not expired. Message 4 is queued because of the periodicity. Although, the $MUT$ was not 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 sporadic message when the $MUT$ expires. This indicates that the periodic transmission of the mixed message cannot be interfered by its sporadic transmission (a unique property of the HCAN protocol). When the event $D$ arrives, a sporadic instance of the mixed message is immediately queued as message 5 because the $MUT$ has already expired. Message 6 is queued due to periodicity.

### 7.2.4 Discussion

In the first method, the Event Timer is reset every time the mixed message is queued for transmission. The implementation of mixed message in method 2 is similar to method 1 to some extent. The main difference is that in method 2, the periodic transmission can be delayed until the expiry of the $MDT$. Whereas in method 1, the periodic transmission is not delayed, in fact, the Event Timer is restarted with every sporadic transmission. The $MDT$ timer is started with every periodic or sporadic transmission of the mixed message. Hence, the worst-case periodicity of the mixed message in methods 1 and 2 can never be higher than the Inhibit Timer and $MDT$ respectively. This means that the models of mixed messages in the first and second implementation methods reduce to the classical sporadic model. Therefore, the existing analyses for CAN messages with offsets [22, 23, 19, 24, 18] can be used for analyzing mixed messages in the first and second implementation methods.

However, the periodic transmission is independent of the sporadic transmission in the third method. The periodic timer is not reset with every sporadic transmission. The mixed message can be queued for transmission even if the $MUT$ is not expired. Hence, the worst-case periodicity of the mixed message is neither bounded by period nor by the $MUT$. Therefore, the analyses in [22, 23, 19, 24, 18] cannot be used for analyzing the mixed messages in the third implementation method.

## 7.3 System Model

The system, $S$, consists of a number of CAN controllers that are connected to a single CAN network. The nodes implement priority-ordered queues, i.e., the highest priority message in a node enters into the bus arbitration. Each CAN message $m_m$ has a unique identifier and a priority denoted by $ID_m$ and $P_m$ respectively. The priority of a message is assumed to be equal to its ID. The priority of $m_m$ is considered higher than the priority of $m_n$ if $P_m < P_n$.

Let the sets $hp(m_m)$, $lp(m_m)$, and $hep(m_m)$ contain the messages with priorities higher, lower, and equal and higher than $m_m$ respectively. Although the priorities of CAN messages are unique, the set $hep(m_m)$ will be used in the case of mixed messages. 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 the standard CAN frame uses an 11-bit identifier whereas the extended CAN frame uses a 29-bit identifier. In order to keep the notations simple and consistent, we define a function $\xi_m$ that

denotes the transmission type of a message. $\xi_m$ specifies whether $m$ is periodic ($P$), sporadic ($S$) or mixed ($M$). Formally, the domain of $\xi_m$ can be defined as follows.

$$\xi_m \in [P, \quad S, \quad M]$$

Each message $m_m$ has a transmission time $C_m$ and queueing jitter $J_m$ which is inherited from the task that queues $m_m$, i.e., the sending task. We assume that $J_m$ can be smaller, equal or greater than $T_m$ or $MUT_m$. 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 and is denoted by $s_m$. In the case of periodic transmission, $m_m$ has a transmission period which is denoted by $T_m$. Whereas in the case of sporadic transmission, $m_m$ has the $MUT_m$ (Minimum Update Time) that refers to the minimum time that should elapse between the transmission of any two sporadic messages. $B_m$ denotes the blocking time of $m_m$ which refers to the largest amount of time $m_m$ can be blocked by any lower priority message.

We duplicate a message when its transmission type is mixed. Hence, each mixed message $m_m$ is treated as two separate messages, i.e., one periodic and the other sporadic. The duplicates share all the attributes except the $T_m$ and $MUT_m$. The periodic copy inherits $T_m$ while the sporadic copy inherits the $MUT_m$. 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). $m_m$ is deemed schedulable if its $R_m$ is less than or equal to its deadline $D_m$. The system is considered schedulable if all of its messages are schedulable. We consider the deadlines to be arbitrary which means that they can be greater than the periods or $MUT$s of corresponding messages. We assume that the CAN controllers are capable of buffering more than one instance of a message. The instances of a message are assumed to be transmitted in the same order in which they are queued (i.e., we assume FIFO policy among the instances of the same message).

Let $O_m$ denotes the offset of $m_m$. We assume that the offset of $m_m$ is always smaller than its period. The first arrival time of $m_m$ is equal to its offset while the subsequent arrivals occur periodically with respect to the first arrival. We assume that the smallest offset in a node is zero. It should be noted that each node has its own local time and there is no global synchronization among the nodes. We assume that the offset relations exist only among periodic messages and periodic copies of mixed messages within a node. We further assume that there are no offset relations:

1. among sporadic messages,
2. between a periodic message and a sporadic message,
3. between a periodic copy of a mixed message and a sporadic message,
4. between the duplicates of a mixed message,
5. between any two messages belonging to different nodes.

All periodic messages and periodic copies of mixed messages in a node are gathered into a single transaction denoted by $\Gamma_i$. Each transaction $\Gamma_i$ belongs to $\Gamma$ which is the set of all transactions in the system. This transactional model is adapted from [31]. It should be noted that the offset relations exist only within a transaction, and there are no offset relations among transactions. In the context of a transaction, we denote a message $m_j$ belonging to transaction $i$ by $m_i^j$. Each transaction has a period denoted by $T_{\Gamma_i}$ and defined as the Least Common Multiple (LCM) of the periods of all messages present in the transaction. Each sporadic message or sporadic copy of a mixed message is modeled as a transaction of its own.

An example of two messages transmitted by a node is depicted in Figure 7.3. $m_1$ is a mixed message with high priority while $m_2$ is a periodic message with low priority. Transaction $\Gamma_1$ contains both $m_2$ and periodic copy of $m_1$. The period of $\Gamma_1$ denoted by $T_{\Gamma_i}$ is the LCM of $T_1$ and $T_2$. Transaction $\Gamma_2$ consists of only the sporadic copy of $m_1$.

## 7.4 Worst-case Response-time Analysis

Let $m_m$ be the message under analysis. We analyze $m_m$ differently based on its transmission type. Intuitively, we consider three different cases namely periodic, sporadic and mixed. We discuss few terms that are used in the analysis. In order to calculate the worst-case response time of $m_m$, the maximum busy period [9, 14] for priority level-m should be known first.

**Maximum Busy Period.** It is the longest contiguous interval of time during which $m_m$ is unable to complete its transmission due to two reasons. First, the bus is occupied by the higher priority messages. Second, a lower priority message already started its transmission when $m_m$ is queued for transmission. The maximum busy period starts at the critical instant.

**Critical Instant.** For a system where messages are scheduled without offsets, the critical instant corresponds to the point in time when all higher priority

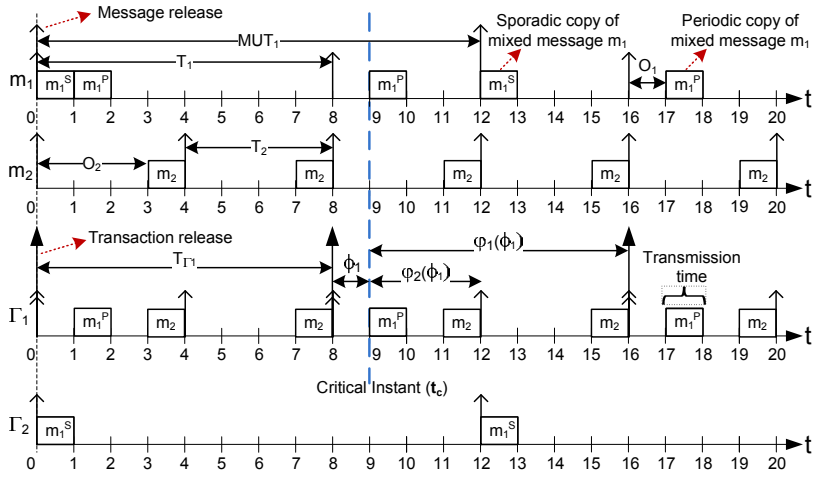


Figure 7.3: Example demonstrating messages and transactions

messages are assumed to be queued simultaneously with $m_m$ while their subsequent instances are assumed to be queued after the shortest possible interval of time [14]. However, this assumption does not hold in the system where messages are scheduled with offsets.

We redefine the critical instant for priority level-m busy period as the instant when (1) $m_m$ or any other higher priority message belonging to the same node as that of $m_m$ is queued for transmission, (2) At least one message with priority higher than $m_m$ is queued for transmission from every node, (3) all sporadic messages and sporadic copies of mixed messages belonging to the set $hp(m_m)$ from every node are simultaneously queued for transmission at the respective nodes, and (4) a lower priority message just started its transmission when $m_m$ is queued. The critical instant for priority level-2 busy period is identified at $t_c$ in Figure 7.3. According to the condition (3) stated above, the arrival of $\Gamma_2$ should coincide with the critical instant.

**Worst-Case Candidates.** The main issue regarding the condition (2) is to determine which message in the set $hp(m_m)$ is the candidate to start the critical instant, i.e., contributing to the worst-case response-time of $m_m$. The solution is that any message in the set $hp(m_m)$ can be the worst-case candidate. Therefore, each message has to be tested in the busy period as the potential worst-case candidate. The response time of $m_m$ should be calculated from ev-

ery worst-case candidate and the maximum among all should be considered as the worst-case response time of $m_m$.

We present the response-time analysis with respect to any worst-case candidate in the following subsections.

### 7.4.1 Case: When $m_m$ is a Periodic Message

Let $m_m$ belongs to transaction $\Gamma_i$. The worst-case response time of $m_m$ is equal to the maximum value among the response times of all of its instances. We calculate the response times of all instances of $m_m$ within the priority level-m busy period. Let $q_m$ be the index variable to denote instances of $m_m$. Let $q_m^L$ and $q_m^H$ denote lowest- and highest-numbered instances respectively. The worst-case response time of $m_m$ is given by:

$$R_m = max\{R_m(q_m)\}, \qquad \forall\, q_m^L \leq q_m \leq q_m^H \tag{7.1}$$

It should be noted that $q_m$ will be equal to 1 if the message instance is queued for transmission between the critical instant and $T_m$. Further, $q_m$ will be equal to 2 if the message instance is queued for transmission between $T_m$ and $2.T_m$. Similarly, $q_m$ will be equal to 0 if the message instance is queued for transmission between the critical instant and $-T_m$. Since the jitter of a message can be greater than its transmission period, it is possible that the previous instances of the message may also be delayed due to jitter and enter in the maximum busy period. The calculations for the response time of instance $q_m$ are adapted from [18] as follows.

$$R_m(q_m) = ST_m + C_m - (\varphi_m(\phi_i) + (q_m - 1).T_m) \tag{7.2}$$

$\phi_i$ in (7.2) denotes the time interval between latest arrival of $\Gamma_i$ (prior to the critical instant) and the critical instant. Consider the example message set in Figure 7.3. $\phi_i$ is equal to 1 time unit and is identified as $\phi_1$ on the third time line from the top. $\varphi_m(\phi_i)$ in (7.2) represents the length of the time interval between the critical instant and first release of $m_m$ that occurs at or after the critical instant. Consider again the example message set in Figure 7.3. $\varphi_m(\phi_i)$ for messages $m_1^P$ and $m_2$ are identified by $\varphi_1(\phi_1)$ and $\varphi_2(\phi_1)$ respectively. The calculations for $\varphi_m(\phi_i)$ are adapted from [31] as follows.

$$\varphi_m(\phi_i) = (T_m - (\phi_i - O_m) \mod T_m) \mod T_m \tag{7.3}$$

$ST_m$ in (7.2) denotes the Start Time (ST) when the priority level-m busy period ends and $m_m(q_m)$ can start its transmission. Basically, it sums up the

interferences due to higher priority messages, previous instances of the same message and the blocking factor. It can be calculated by solving the following equation.

$$ST_m^{n+1} = B_m + (q_m - q_m^L).C_m + \sum_{\forall \Gamma_k \in \Gamma} W_m(\Gamma_k, \phi_k, ST_m^n) \tag{7.4}$$

Where the term $(q_m - q_m^L).C_m$ represents the interference due to previous instances of $m_m$ that are queued ahead of the instance under analysis. $B_m$ is the blocking delay for $m_m$. It is defined as the amount of time equal to the largest transmission time in the set of lower priority messages. $B_m$ is calculated as follows.

$$B_m = \max_{\forall m_j \in lp(m_m)} \{C_j\} \tag{7.5}$$

(7.4) is an iterative equation. It is solved iteratively until two consecutive solutions become equal. The starting value for $ST_m^n$ in (7.4) can be selected equal to $B_m + (q_m - q_m^L).C_m$. In (7.4), $W_m$ represents the amount of interference due to the messages in the set $hp(m_m)$ that are queued for transmission since the beginning of the busy period. It is important to mention that a message cannot be interfered by higher priority messages during its transmission because CAN uses fixed-priority non-preemptive scheduling. Whenever we use the term interference, it refers to the amount of time $m_m$ has to wait in the send queue because the higher priority messages win the arbitration, i.e., the right to transmit before $m_m$. $W_m$ can be calculated as follows.

$$W_m(\Gamma_k, \phi_k, ST_m^n) = \sum_{\forall m_j \in hp_k(m_m)} \Upsilon_k^j(ST_m^n).C_j \tag{7.6}$$

Where $hp_k(m_m)$ represents the set of all those messages that belong to $\Gamma_k$ and have priority higher than $m_m$. $\Upsilon_k^j(ST_m^n)$ in (7.6) is calculated differently based on the transmission type $\xi_j$ of the higher priority message $m_j$. The calculations for $\Upsilon_k^j(ST_m^n)$ are adapted from [18] and [16] as follows. It should be noted that the existing analysis considers only higher priority periodic messages while calculating $\Upsilon_k^j(ST_m^n)$. On the other hand, we consider all higher

priority periodic, sporadic and mixed messages while calculating $\Upsilon_k^j(ST_m^n)$.

$$\Upsilon_k^j(ST_m^n) = \begin{cases} \left\lfloor \frac{J_j+\varphi_j(\phi_k)}{T_j} \right\rfloor + \left\lfloor \frac{ST_m^n-\varphi_j(\phi_k)}{T_j} \right\rfloor + 1, & \text{if } \xi_j = \text{P} \\ \left\lfloor \frac{ST_m^n+J_j}{MUT_j} \right\rfloor + 1, & \text{if } \xi_j = \text{S} \\ \left\lfloor \frac{J_j+\varphi_j(\phi_k)}{T_j} \right\rfloor + \left\lfloor \frac{ST_m^n-\varphi_j(\phi_k)}{T_j} \right\rfloor + 1 & \\ \quad + \left\lfloor \frac{ST_m^n+J_j}{MUT_j} \right\rfloor + 1, & \text{if } \xi_j = \text{M} \end{cases} \tag{7.7}$$

Where $\varphi_j(\phi_k)$ is calculated by replacing the indices $_m$ and $_i$ with $_j$ and $_k$ in (7.3) respectively. In (7.7), the periodic case is adapted from [18]. $\left\lfloor \frac{J_j+\varphi_j(\phi_k)}{T_j} \right\rfloor$ represents the maximum number of instances of the higher priority periodic message or periodic copy of mixed message $m_j$ that may accumulate at the critical instant. Whereas $\left\lfloor \frac{ST_m^n-\varphi_j(\phi_k)}{T_j} \right\rfloor + 1$ represents the maximum number of instances of $m_j$ that are queued for transmission in the interval that starts with the critical instance and ends at $\Upsilon_m^n$.

There are no offset relations of $m_m$ with any sporadic message. Moreover, all sporadic messages are assumed to be queued for transmission at the critical instant. Therefore, the sporadic and mixed cases in (7.7) are adapted from [16]. $\left\lfloor \frac{ST_m^n+J_j}{MUT_j} \right\rfloor + 1$ represent the maximum number of instances of higher priority sporadic message or sporadic copy of mixed message $m_j$ that are queued for transmission in the interval that starts with the critical instance and ends at $\Upsilon_m^n$. This also includes the number of instances of $m_j$ that may accumulate at the critical instant due to jitter.

It is evident from (7.7) that interference from both periodic and sporadic copies of every higher priority mixed message is taken into account. The lowest- and highest-numbered instances of $m_m$ denoted by $q_m^L$ and $q_m^H$ are calculated as follows.

$$q_m^L = -\left\lfloor \frac{J_m + \varphi_m(\phi_i)}{T_m} \right\rfloor + 1 \tag{7.8}$$

$$q_m^H = \left\lceil \frac{L_m - \varphi_m(\phi_i)}{T_m} \right\rceil \tag{7.9}$$

Where $L_m$ represents the length of priority level-m busy period. The existing analysis [18] considers only higher priority periodic messages while calculating $L_m$. We adapt the existing analysis to consider all higher priority periodic, sporadic and mixed messages while calculating $L_m$. Similar to (7.4), $L_m$ can be calculated using the fixed-point method as follows.

$$L_m^{n+1} = \left[\left\lfloor \frac{J_m + \varphi_m(\phi_i)}{T_m} \right\rfloor + \left\lceil \frac{L_m^n - \varphi_m(\phi_i)}{T_m} \right\rceil\right].C_m + B_m + \sum_{\forall \Gamma_k \in \Gamma, m_j \in hp_k(m_m)} \mathsf{M}_k^j(L_m^n).C_j \tag{7.10}$$

Where

$$\mathsf{M}_k^j(L_m^n) = \begin{cases} \left\lfloor \frac{J_j+\varphi_j(\phi_k)}{T_j} \right\rfloor + \left\lceil \frac{L_m^n-\varphi_j(\phi_k)}{T_j} \right\rceil, & \text{if } \xi_j = \text{P} \\ \left\lfloor \frac{L_m^n+J_j}{MUT_j} \right\rfloor + 1, & \text{if } \xi_j = \text{S} \\ \left\lfloor \frac{J_j+\varphi_j(\phi_k)}{T_j} \right\rfloor + \left\lceil \frac{L_m^n-\varphi_j(\phi_k)}{T_j} \right\rceil & \\ + \left\lfloor \frac{L_m^n+J_j}{MUT_j} \right\rfloor + 1, & \text{if } \xi_j = \text{M} \end{cases} \tag{7.11}$$

### 7.4.2 Case: When $m_m$ is a Sporadic Message

Let again the message under analysis be $m_m$ that belongs to $\Gamma_i$, i.e, the transaction of its own. The worst-case response time of $m_m$ can be calculated similar to the periodic case with one exception. That is, sporadic message does not hold any offset relations with any other message in the system. Moreover, all sporadic messages including $m_m$ are assumed to be queued for transmission at the critical instant. Intuitively, $\phi_i$ will be equal to $MUT_m$, i.e., the latest arrival of $m_m$ prior to critical instant will be $MUT_m$ time units before the critical instant. Let us use $O_m$ equal to zero, and $MUT_m$ in place of both $T_m$ and $\phi_i$ in (7.3).

$$\varphi_m(\phi_i) = 0 \tag{7.12}$$

In this case, (7.1), (7.4), (7.5), (7.6), (7.7) and (7.11) hold intact. However, we need to replace the new value of $\varphi_m(\phi_i)$ from (7.12) in the calculations for (7.2), (7.8), (7.9) and (7.10) as follows.

$$R_m(q_m) = ST_m + C_m - (q_m - 1).MUT_m \tag{7.13}$$

$$q_m^L = -\left\lfloor \frac{J_m}{MUT_m} \right\rfloor + 1 \tag{7.14}$$

$$q_m^H = \left\lceil \frac{L_m}{MUT_m} \right\rceil \tag{7.15}$$

$$L_m^{n+1} = \left[ \left\lfloor \frac{J_m}{MUT_m} \right\rfloor + \left\lceil \frac{L_m^n}{MUT_m} \right\rceil \right].C_m + B_m + \sum_{\forall \Gamma_k \in \Gamma, m_j \in hp_k(m_m)} \mathsf{M}_k^j(L_m^n).C_j \tag{7.16}$$

### 7.4.3 Case: When $m_m$ is a Mixed Message

Let again $m_m$ be the message under analysis. Since a mixed message is duplicated as two separate messages, the extended analysis treats them separately. Let the periodic and sporadic copies of $m_m$ be denoted by $m_{m_P}$ and $m_{m_S}$ respectively. We denote the worst-case response times of $m_{m_P}$ and $m_{m_S}$ by $R_{m_P}$ and $R_{m_S}$ respectively. The worst-case response time of $m_m$ is the maximum between $R_{m_P}$ and $R_{m_S}$ as follows.

$$R_m = max\{R_{m_P}, \ \ R_{m_S}\} \tag{7.17}$$

Where $R_{m_P}$ and $R_{m_S}$ are equal to the maximum value among the response times of their respective instances. Let $q_{m_P}$ be the index variable to denote the instances of $m_{m_P}$. Let $q_{m_P}^L$ and $q_{m_P}^H$ denote the lowest- and highest-numbered instances of $m_{m_P}$ respectively. Let $q_{m_S}$, $q_{m_S}^L$ and $q_{m_S}^H$ denote the index variable for instances, and lowest- and highest-numbered instances of $m_{m_S}$ respectively. The calculations for $R_{m_P}$ and $R_{m_S}$ are adapted from the periodic and sporadic cases respectively as follows.

$$R_{m_P} = max\{R_{m_P}(q_{m_P})\}, \forall \ q_{m_P}^L \leq q_{m_P} \leq q_{m_P}^H \tag{7.18}$$

$$R_{m_S} = max\{R_{m_S}(q_{m_S})\}, \forall \ q_{m_S}^L \leq q_{m_S} \leq q_{m_S}^H \tag{7.19}$$

The calculations for the worst-case response time of each instance of $m_{m_P}$ and $m_{m_S}$ are adapted from (7.2) and (7.13) as follows.

$$R_{m_P}(q_{m_P}) = ST_{m_P} + C_m - (\varphi_{m_P}(\phi_i) + (q_{m_P} - 1).T_m) \tag{7.20}$$

$$R_{m_S}(q_{m_S}) = ST_{m_S} + C_m - (q_{m_S} - 1).MUT_m \tag{7.21}$$

Where $\varphi_{m_P}(\phi_i)$ is calculated using (7.3). The calculations for $ST_{m_P}$ and $ST_{m_S}$ are adapted from (7.4) after some augmentation.

$$ST^{n+1}_{m_P} = B_m + Q^P_{m_S}.C_m + (q_{m_P} - q^L_{m_P}).C_m + \sum_{\forall \Gamma_k \in \Gamma} W_{m_P}(\Gamma_k, \phi_k, ST^n_{m_P}) \tag{7.22}$$

$$ST^{n+1}_{m_S} = B_m + Q^S_{m_P}.C_m + (q_{m_S} - q^L_{m_S}).C_m + \sum_{\forall \Gamma_k \in \Gamma} W_{m_S}(\Gamma_k, \phi_k, ST^n_{m_S}) \tag{7.23}$$

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

$Q^P_{m_S}.C_m$ and $Q^S_{m_P}.C_m$ in (7.22) and (7.23) respectively represent the effect of self interference in a mixed message. By self interference we mean that the periodic copy of a mixed message can be interfered by the sporadic copy and vice versa. Since, both $m_{m_P}$ and $m_{m_S}$ have equal priorities, any instance of $m_{m_S}$ queued ahead of $m_{m_P}$ will contribute an extra delay to the worst-case queueing delay experienced by $m_{m_P}$. A similar argument holds in the case of $m_{m_S}$. We reuse the effect of self interference in a mixed message that we derived in [16] as follows.

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 7.4. In the first case, we assume $T_m$ to be greater than $MUT_m$. That is, there can be more transmissions of $m_{m_S}$ compared to that of $m_{m_P}$. Since, the maximum update time between the queueing of any two instances of $m_{m_S}$ can be arbitrarily very long, it is possible to have fewer sporadic transmissions than periodic transmissions of $m$. In the second case, we assume that $T_m$ is equal to $MUT_m$. In this case, there are equal number of transmissions of $m_{m_P}$ and $m_{m_S}$. In the third case, we assume that $T_m$ is smaller than $MUT_m$. This implies that the number of sporadic transmissions will be less than the periodic transmissions of $m$.

It is important to note that in the example shown in Figure 7.4, there is a small offset between the first periodic and sporadic transmission of $m$. This offset is used to maximize the queueing delay. If this offset is removed then only one message will be queued corresponding to the first instance of both $m_{m_P}$ and $m_{m_S}$. 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 sporadic transmissions of $m$ will not overlap, there by, maximizing the queueing delay.

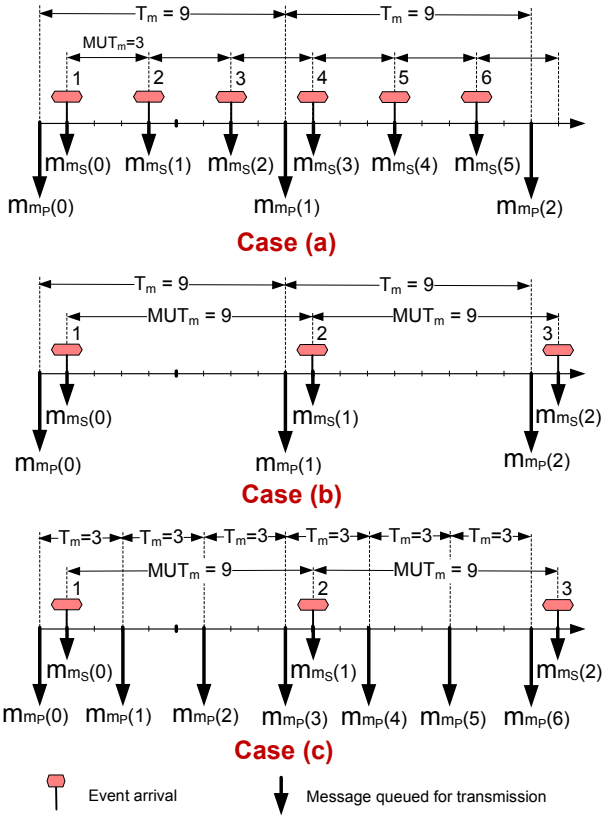


Figure 7.4: Self interference in a mixed message: (a) $\mathbf{T_m > MUT_m}$, (b) $\mathbf{T_m = MUT_m}$, (c) $\mathbf{T_m < MUT_m}$

.

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

Let the message under analysis be $m_{m_P}$ and consider case (a) in Figure 7.4. An application task queues $m$ periodically with a period $T_m$ (equal to 9 time units). Moreover, the same task can also queue $m$ sporadically at the arrival of events (labeled with numbers 1-6). The queueing of $m_{m_E}$ is constrained by

$MUT_m$ (equal to 3 time units). The first instance of $m_{m_P}$ ($q_{m_P} = 0$) is queued for transmission as shown by $m_{m_P}(0)$ in Figure 7.4. If event 1 had arrived at the same time as the queueing of $m_{m_P}(0)$ then the signals in $m_{m_S}(0)$ would have been updated as part of $m_{m_P}(0)$. In that case, $m_{m_S}(0)$ would not have been queued separately (this is the property of the mixed message in the HCAN protocol). In order to maximize the contribution of $m_{m_S}$ on the queueing delay of $m_{m_P}$, $m_{m_S}(0)$ is queued just after the queueing of $m_{m_P}(0)$ as shown in all the cases in Figure 7.4. Therefore, $m_{m_S}(0)$ and subsequent instances of $m_{m_S}$ will have no contribution in the worst-case queueing delay of the first instance of $m_{m_P}$ denoted by $m_{m_P}(0)$.

Consider the second instance of $m_{m_P}$. All the instances of $m_{m_S}$ that are queued ahead of $m_{m_P}(1)$ will contribute to its worst-case queueing delay. It can be observed in the case (a) that the first three instances of $m_{m_S}$ are queued ahead of $m_{m_P}(1)$. Similarly, there are six instances of $m_{m_S}$ that are queued ahead of $m_{m_P}(2)$.

Let $Q^P_{m_S}$ denotes the total number of instances of $m_{m_S}$ that are queued ahead of the $q^{th}_{m_P}$ instance of $m_{m_P}$. We can generalize $Q^P_{m_S}$ for the case (a) as follows.

$$Q^P_{m_S} = \left\lceil \frac{q_{m_P} T_m}{MUT_m} \right\rceil \tag{7.24}$$

For example, consider again the queueing of different instances of $m_{m_S}$ and $m_{m_P}$ in the case (a). Equation (7.24) yields the set $\{Q^P_{m_S} = 0, 3, 6, ...\}$ for the corresponding values in the set $\{q_{m_P} = 0, 1, 2, ...\}$. Thus the total number of instances of $m_{m_S}$ queued ahead of each instance of $m_{m_p}$ calculated by (7.24) are consistent with the case (a) in Figure 7.4.

**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 7.4 that there are 0, 1, and 2 instances of $m_{m_S}$ that are queued ahead of $m_{m_P}(0)$, $m_{m_P}(1)$ and $m_{m_P}(2)$ respectively. When Equation (7.24) is used in case (b), we get the set $\{Q^P_{m_S} = 0, 1, 2, ...\}$ for the corresponding values in the set $\{q_{m_P} = 0, 1, 2, ...\}$. Therefore, (7.24) is also applicable on case (b).

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

Now, consider case (c) in which $T_m$ (3 time units) is smaller than $MUT_m$ (9 time units). The first instance of $m_{m_S}$ denoted by $m_{m_S}(0)$ will be queued ahead of $m_{m_P}(1)$, $m_{m_P}(2)$ and $m_{m_P}(3)$. Similarly, the two instances of $m_{m_S}$ denoted by $m_{m_S}(0)$ and $m_{m_S}(1)$ will contribute to the worst-case queueing delay of $m_{m_P}(4)$, $m_{m_P}(5)$ and $m_{m_P}(6)$. (7.24) yields the set $\{Q^P_{m_S} = 0, 1, 1,$

$1, 2, 2, 2, ...\}$ for the corresponding values in the set $\{q_{m_P} = 0, 1, 2, 3, 4, 5, 6, ...\}$. Thus the total number of instances of $m_{m_S}$ queued ahead of each instance of $m_{m_P}$ calculated by equation (7.24) are consistent with the case (c) in Figure 7.4.

Now we consider the effect of jitter on the instances of $m_{m_S}$ prior to $m_{m_S}(0)$ which can be queued just ahead of $m_{m_P}(0)$ and contribute to the worst-case queueing delay of $m_{m_P}$. We assume FIFO queueing policy among the instances of the same message. Due to the offset of $m_{m_P}$, there may be one or more instances of $m_{m_S}$ that can be queued ahead of the first instance of $m_{m_P}$. Hence, the offset of $m_{m_P}$ should also be taken into account when considering the self interference from $m_{m_S}$. By considering the jitter of $m_{m_S}$ and offset of $m_{m_P}$ to $Q^P_{m_S}$, equation (7.24) can be generalized for the three cases as follows.

$$Q^P_{m_S} = \left\lceil \frac{q_{m_P} T_m + J_m + O_m}{MUT_m} \right\rceil \tag{7.25}$$

The total number of instances of $m_{m_P}$ that are queued ahead of the $q^{th}_{m_S}$ instance of $m_{m_S}$, denoted by $Q^S_{m_P}$, can be derived in a similar fashion. However, $Q^S_{m_P}$ does not contain the term $O_m$ because sporadic messages do not have any offset relations with any other message. Thus $Q^S_{m_P}$ can be calculated by the following equation.

$$Q^S_{m_P} = \left\lceil \frac{q_{m_S} MUT_m + J_m}{T_m} \right\rceil \tag{7.26}$$

Using the values of $Q^S_{m_P}$ and $Q^P_{m_S}$ from (7.26) and (7.25) in (7.22) and (7.23) respectively as follows.

$$\begin{aligned} ST^{n+1}_{m_P} &= B_m + \left\lceil \frac{q_{m_P}.T_m + J_m + O_m}{MUT_m} \right\rceil .C_m \\ &+ (q_{m_P} - q^L_{m_P}).C_m + \sum_{\forall \Gamma_k \in \Gamma} W_{m_P}(\Gamma_k, \phi_k, ST^n_{m_P}) \end{aligned} \tag{7.27}$$

$$\begin{aligned} ST^{n+1}_{m_S} &= B_m + \left\lceil \frac{q_{m_S}.MUT_m + J_m}{T_m} \right\rceil .C_m \\ &+ (q_{m_S} - q^L_{m_S}).C_m + \sum_{\forall \Gamma_k \in \Gamma} W_{m_S}(\Gamma_k, \phi_k, ST^n_{m_S}) \end{aligned} \tag{7.28}$$

The calculations for $W_{m_P}$, $q^L_{m_P}$, $q^H_{m_P}$ and $L_{m_P}$ are done using (7.6), (7.8), (7.9) and (7.10) by replacing the index $_m$ with $_{m_P}$ respectively. Similarly,

| ECU | $m_m$ | $P_m$ | $\xi_m$ | $T_m$ | $MUT_m$ | $C_m$ | $O_m$ | $J_m$ | $D_m$ | $R_m$ |
|---|---|---|---|---|---|---|---|---|---|---|
| $ECU_S$ | $m_1$ | 1 | $P$ | 10 | - | 1 | 0 | 1 | 10 | 3 |
| $ECU_S$ | $m_2$ | 2 | $M$ | 10 | 10 | 1 | 1 | 11 | 20 | 15 |
| $ECU_S$ | $m_3$ | 3 | $S$ | - | 10 | 1 | 0 | 0 | 10 | 7 |
| $ECU_W$ | $m_4$ | 4 | $P$ | 10 | - | 1 | 0 | 0 | 10 | 8 |
| $ECU_W$ | $m_5$ | 5 | $P$ | 10 | - | 1 | 2 | 12 | 20 | 18 |

Table 7.1: Attributes and response times of periodic, sporadic and mixed messages in the steer-by-wire system

$W_{m_S}$, $q^L_{m_S}$, $q^H_{m_S}$ and $L_{m_S}$ are calculated using (7.6), (7.14), (7.15) and (7.16) by replacing the index $_m$ with $_{m_S}$ respectively. Further, the calculations in (7.5), (7.7) and (7.11) hold intact with proper replacement of the index variable for both $m_{m_P}$ and $m_{m_S}$.

## 7.5 Automotive-application Case Study

In this section, we conduct the Steer-By-Wire (SBW) case study. The SBW system is an automotive feature that substitutes most of the mechanical and hydraulic components with the electronic components in the steering system of the vehicle. We adapt the SBW system from [32]. The partial architecture of the SBW system is shown in Figure 7.5. There are two ECUs (rest of the ECUs are not shown for simplicity) that are connected to the CAN network.

The Steering Control ECU denoted by $ECU_S$ receives input from three sensors that correspond to the steering angle, steering torque (applied by the driver) and vehicle speed signals. It sends three messages $m_1$, $m_2$ and $m_3$ to the network. These messages carry information regarding steering angle, torque and feedback signals. $m_1$ is a periodic, $m_2$ is a mixed and $m_3$ is a sporadic message. $ECU_S$ receives the periodic message $m_4$ that contains information about wheel torque that is sent by the Wheel Control ECU. Based on these inputs, it calculates the feedback steering torque and sends it to the feedback actuator. The feedback torque actuator is responsible for producing the turning effect of the steering which in turn produces the feeling of turning the wheels for the driver.

Similarly, the Wheel Control ECU denoted by $ECU_W$ acquires signals from wheel angle and wheel torque sensors. Depending upon these signals and the signals received in the CAN message, it calculates the wheel torque, and

produces actuation signals for the wheel control actuators. Apart from $m_4$, it also sends a periodic message $m_5$ to the network. The timing attributes of all messages are shown in Table 1. We analyzed this message set with the extended analysis. The response times calculated using the extended analysis are also listed in Table 1. Since, the jitter of messages $m_2$ and $m_5$ is higher than the corresponding periods, there are several instances of these messages present in the corresponding busy periods. For example, there are four instances of $m_2$ that are present in the priority level-2 busy period. The extended analysis calculates the response times of all these instances and considers maximum among them as the worst-case response time of $m_2$. It should be noted that correct analysis of this message set would not have been possible with the existing analysis because it contains a mixed message whose jitter and deadline are greater than the corresponding period.

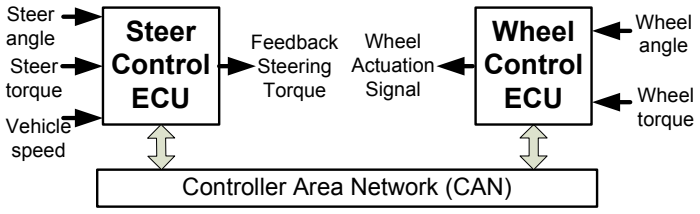


Figure 7.5: Partial architecture of the steer-by-wire system

## 7.6 Conclusion

The existing response-time analysis of CAN does not support the analysis of mixed messages that are scheduled with offsets and have jitter and deadlines higher than their transmission periods. Message jitter can be higher than its period in practical systems. We extended the existing offset-based analysis for CAN by lifting the restrictions on message jitter and deadline. The extended analysis provides safe upper bounds on the response times of mixed messages that are scheduled with offsets. Mixed messages are implemented by several higher-level protocols for CAN that are used in the automotive industry today. The extended analysis is applicable to any higher-level protocol for CAN that uses periodic, sporadic, and mixed transmission of messages that are scheduled with offsets. We also conducted the automotive-application case study to demonstrate the applicability of our analysis.

In the future, we plan to develop an optimized offset assignment method for the systems that contain periodic as well as mixed messages. We also plan to implement the extended analysis as a plug-in for the existing industrial tool suite the Rubus-ICE [13].

## Acknowledgement

The work in this paper is supported by the Swedish Knowledge Foundation (KKS) within the projects SythSoft and FEMMVA. The authors would like to thank the industrial partners Arcticus Systems AB, BAE Systems Hägglunds and Volvo Construction Equipment (VCE), Sweden.

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

[4] M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, 1986.

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

[6] 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.

[7] 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.

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

[9] 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.

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

[11] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Support for Holistic Response-time Analysis in an Industrial Tool Suite: Implementation Issues, Experiences and a Case Study. In *19th IEEE Conference on Engineering of Computer Based Systems (ECBS)*, pages 210 –221, April 2012.

[12] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Support for end-to-end response-time and delay analysis in the industrial tool suite: Issues, experiences and a case study. *Computer Science and Information Systems, ISSN: 1361-1384*, 10(1), 2013.

[13] Rubus-ICE. http://www.arcticus-systems.com.

[14] 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.

[15] 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*, July 2011.

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

[17] S. Mubeen, J. Mäki-Turja and M. Sjödin. Response-Time Analysis of Mixed Messages in Controller Area Network with Priority- and FIFO-Queued Nodes. In *9th IEEE International Workshop on Factory Communication Systems (WFCS)*, May 2012.

[18] Patrick Yomsi, Dominique Bertrand, Nicolas Navet, and Robert Davis. Controller Area Network (CAN): Response Time Analysis with Offsets. In *9th IEEE International Workshop on Factory Communication Systems (WFCS)*, May 2012.

[19] Alexander Szakaly. Response Time Analysis with Offsets for CAN. Master's thesis, Department of Computer Engineering, Chalmers University of Technology, Nov. 2003.

[20] Mathieu Grenier, Lionel Havet, and Nicolas Navet. Pushing the limits of can- scheduling frames with offsets provides a major performance boost. In *4th European Congress on Embedded Real Time Software (ERTS)*, 2008.

[21] Mathieu Grenier, Lionel Havet, and Nicolas Navet. Automotive embedded systems handbook. In Nicolas Navet and Francoise Siminot-Lion, editors, *Chapter 14: Scheduling messages with offsets on Controller Area Network - a major performance boost*. CRC Press, 2009.

[22] Yang Chen, Ryo Kurachi, Hiroaki Takada, and Gang Zeng. Schedulability comparison for can message with offset: Priority queue versus fifo queue. In *19th International Conference on Real-Time and Network Systems (RTNS)*, pages 181–192, Sep. 2011.

[23] Lei Du and Guoqing Xu. Worst case response time analysis for can messages with offsets. In *IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, pages 41 –45, nov. 2009.

[24] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst. System level performance analysis - the symta/s approach. *Computers and Digital Techniques*, 152(2):148–166, March 2005.

[25] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Worst-case response-time analysis for mixed messages with offsets in controller area network. In *17th IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, sept. 2012.

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

[27] CANopen Application Layer and Communication Profile. CiA Draft Standard 301. Version 4.02. February 13, 2002. http://www.can-cia.org/index.php?id=440.

[28] Requirements on Communication, Release 3.0, Rev 7, Ver. 2.2.0. The AUTOSAR Consortium, Sep., 2010. www.autosar.-org/download/R3.0/AUTOSAR_SRS_COM.pdf.

[29] AUTOSAR Techincal Overview, Version 2.2.2., Release 3.1, The AUTOSAR Consortium, Aug., 2008. http://autosar.org.

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

[31] J.C. Palencia and M. Gonzalez Harbour. Schedulability Analysis for Tasks with Static and Dynamic Offsets. *IEEE International Symposium on Real-Time Systems (RTSS)*, 1998.

[32] Khaled Chaaban, Patrick Leserf, and Sébastien Saudrais. Steer-by-wire system development using AUTOSAR methodology. In *14th IEEE Conference on Emerging Technologies and Factory Automation*, 2009.

# Chapter 8

# Paper E:
# MPS-CAN Analyzer: Integrated Implementation of Response-Time Analyses for Controller Area Network

Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin
In Journal of Systems Architecture (JSA), Elsevier, May, 2014.

**Abstract**

We present a new response-time analyzer for Controller Area Network (CAN) that integrates and implements a number of response-time analyses which address various transmission modes and practical limitations in the CAN controllers. The existing tools for the response-time analysis of CAN support only periodic and sporadic messages. They do not analyze mixed messages which are partly periodic and partly sporadic. These messages are implemented by several higher-level protocols based on CAN that are used in the automotive industry. The new analyzer supports periodic, sporadic as well as mixed messages. It can analyze the systems where periodic and mixed messages are scheduled with offsets. It also supports the analysis of all types of messages while taking into account several queueing policies and buffer limitations in the CAN controllers such as abortable or non-abortable transmit buffers. Moreover, the tool supports the analysis of mixed, periodic and sporadic messages in the heterogeneous systems where Electronic Control Units (ECUs) implement different types of queueing policies and have different types of buffer limitations in the CAN controllers. We conduct a case study of a heterogeneous application from the automotive domain to show the usability of the tool. Moreover, we perform a detailed evaluation of the implemented analyses.

## 8.1 Introduction

The Controller Area Network (CAN) [1] is a widely used real-time network protocol in the automotive domain. In 2003, it was standardized by the International Organization for Standardization in ISO 11898-1 [2]. It is a multi-master, event-triggered, serial communication protocol supporting bus speeds of up to 1 megabits per second. Over 850 million CAN enabled controllers were sold in 2011 according to the CAN in Automation (CiA) [3] estimate. Over 2 billion controllers have been sold to date and most of them have been used in the automotive industry. The CAN protocol also finds its applications in other domains, e.g., industrial control, medical equipments, maritime electronics, and production machinery. There are several higher-level protocols for CAN that are developed for many industrial applications such as CAN Application Layer (CAL), CANopen, J1939, Hägglunds Controller Area Network (HCAN), and CAN for Military Land Systems domain (MilCAN).

Often, CAN is used in hard real-time systems. The providers of these systems are required to ensure that the systems meet their deadlines. In order to provide evidence that each action by the system will be provided in a timely manner, *a priori* analysis techniques, such as schedulability analysis [4, 5, 6], have been developed by the research community. Response-Time Analysis (RTA) [4, 5, 6, 7] 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 network respectively.

### 8.1.1 Paper Contribution

There is a limitation with the existing response-time analyses for CAN and the corresponding tools that implement these analyses. That is, they support only periodic and sporadic messages. They do not support the analysis of mixed messages which are partly periodic and partly sporadic. Mixed messages are simultaneously time- and event-triggered and are implemented by several higher-level protocols based on CAN that are used in the automotive industry today. To the best our knowledge, there is no freely-available tool that implements the analysis of mixed messages (a commercial tool Rubus-ICE implements basic analysis of mixed messages in CAN). In this paper we present a new response-time analyzer for CAN namely MPS-CAN analyzer (MPS stands for Mixed, Periodic and Sporadic). It supports the analysis of periodic, sporadic and mixed messages. It implements several extensions of RTA for CAN taking into account the following aspects:

- analysis of mixed messages;
- analysis of messages scheduled with or without offsets;
- analysis of messages having arbitrary jitter and deadlines;
- analysis of network with CAN controllers implementing different queueing policies, e.g., priority or First-In, First-Out (FIFO),
- analysis of network with no buffer limitations in the CAN controllers, i.e., the controllers implement such a large (but finite) number of transmit buffers that there is no need to abort transmission requests;
- analysis of network with limitations in CAN controllers, e.g., the controllers implement abortable or non-abortable transmit buffers.

The tool also supports the analysis of mixed, periodic and sporadic messages in heterogeneous systems where Electronic Control Units (ECUs) implement different types of queueing policies and have different types of buffer limitations in the CAN controllers. In these systems, the tool treats each message differently depending upon its transmission type, and the type of queueing policy and buffer limitations in the sender ECU. We also conduct a case study in which we analyze the CAN messages in the heterogeneous system to show usability of the tool. Moreover, we perform a detailed evaluation of the implemented analyses.

### 8.1.2 Paper Layout

The remainder of the paper is organized as follows. In Section 8.2, we discuss mixed transmission patterns supported by several higher-level protocols. In Section 8.3, we discuss the practical limitations in the CAN controllers. Section 8.4 discusses the related works. Section 8.5 discusses the implemented analyses, layout and usability of the MPS-CAN analyzer. Section 8.6 presents the case study and evaluation. Finally, Section 8.7 concludes the paper.

## 8.2 Mixed Transmission Supported by Higher-level Protocols

The analysis implemented in the MPS-CAN analyzer supports periodic and sporadic as well as mixed messages. In this section, we discuss and compare

the implementation of mixed messages by several higher-level protocols for CAN. Traditionally, it is assumed that the tasks queueing CAN messages are invoked either by periodic or sporadic events. If a message is queued for transmission at periodic intervals, we use the term "Period" to refer to its periodicity. A sporadic message is queued for transmission as soon as an event occurs that changes the value of one or more signals contained in the message provided the Minimum Update Time ($MUT$[1]) between the queueing of two successive sporadic messages has elapsed. However, there are some higher-level protocols and commercial extensions of CAN in which the tasks that queue the messages can be invoked periodically as well as sporadically. If a message can be queued for transmission periodically as well as sporadically, it is said to be mixed. In other words, a mixed message is simultaneously time- and event-triggered. We identified three different types of implementations of mixed messages used in the industry.

### 8.2.1 Method 1: Implementation in the CANopen Protocol

The CANopen protocol [8] supports mixed transmission that corresponds to the Asynchronous Transmission Mode coupled with the Event Timer. The Event Timer is used to transmit an asynchronous message cyclically. A mixed message can be queued for transmission at the arrival of an event provided the 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 when the Event Timer expires. The Event Timer is reset every time the message is queued. Once a mixed message is queued, any additional queueing of this message will not take place during the Inhibit Time [8]. The transmission pattern of a mixed message in CANopen is illustrated in Figure 8.1(a). The down-pointing arrows symbolize the queueing of messages while the upward lines (labeled with alphabetic characters) represent arrival of the events. Message $1$ is queued as soon as the event $A$ arrives. Both the Event Timer and Inhibit Time are reset. As soon as the Event Timer expires, message $2$ is queued due to periodicity and both the Event Timer and Inhibit Time are reset again. When the event $B$ arrives, message $3$ is immediately queued because the Inhibit Time has already expired. Note that the Event Timer is also reset at the same time when message $3$ is queued as shown in Figure 8.1(a). Message $4$ is queued because of the expiry of the Event Timer. There exists a dependency relationship between the Inhibit

[1] We overload the term "$MUT$" to refer to the Inhibit Time in the CANopen protocol and the Minimum Delay Time (MDT) in the AUTOSAR communication.

Time and the Event Timer, i.e., the Event Timer is reset with every sporadic transmission.

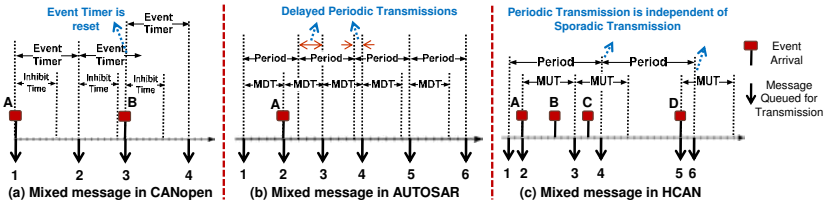


Figure 8.1: Mixed transmission pattern in higher-level protocols for CAN

### 8.2.2 Method 2: Implementation in the AUTOSAR Communications

AUTOSAR (AUTomotive Open System ARchitecture) [9] can be viewed as a higher-level protocol if it uses CAN for network communication. Mixed transmission mode in AUTOSAR is widely used in practice. In AUTOSAR, a mixed message can be queued for transmission repeatedly with a period equal to the mixed transmission mode time period. The mixed message can also be queued at the arrival of an event provided the Minimum Delay Time ($MDT$) has been expired. However, each transmission of a mixed message, regardless of being periodic or sporadic, is limited by the $MDT$. This means that both periodic and sporadic transmissions are delayed until the $MDT$ expires. The transmission pattern of a mixed message implemented by AUTOSAR is illustrated in Figure 8.1(b). Message $1$ is queued (the $MDT$ is started) because of partly periodic nature of a mixed message. When the event $A$ arrives, message $2$ is queued immediately because the $MDT$ has already expired. The next periodic transmission is scheduled $2$ time units after the transmission of message $2$. However, the next two periodic transmissions corresponding to messages $3$ and $4$ are delayed because the $MDT$ is not expired. This is indicated by the text "Delayed Periodic Transmissions" in Figure 8.1(b). The periodic transmissions corresponding to messages $5$ and $6$ take place at the scheduled times because the $MDT$ is already expired in both cases.

### 8.2.3 Method 3: Implementation in the HCAN Protocol

A mixed message in the HCAN protocol [10] contains signals out of which some are periodic and some are sporadic. 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 the $MUT$ between the queueing of two successive sporadic instances of the mixed message has elapsed. Hence, the transmission of the mixed message due to arrival of events is constrained by the $MUT$. The transmission pattern of the mixed message is illustrated in Figure 8.1(c). Message 1 is queued because of periodicity. As soon as event $A$ arrives, message 2 is queued. When event $B$ arrives it is not queued immediately because the $MUT$ is not expired yet. As soon as the $MUT$ expires, message 3 is queued. Message 3 contains the signal changes that correspond to event $B$. Similarly, a message is not immediately queued when the event $C$ arrives because the $MUT$ is not expired. Message 4 is queued because of the periodicity. Although, the $MUT$ was not 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 sporadic message when the $MUT$ expires. This indicates that the periodic transmission of a mixed message cannot be interfered by its sporadic transmission. This is a unique property of the HCAN protocol. When the event $D$ arrives, a sporadic instance of the mixed message is immediately queued as message 5 because the $MUT$ has already expired. Message 6 is queued due to the partly periodic nature of the mixed message.

### 8.2.4 Discussion

In the first method [8], the Event Timer is reset every time the mixed message is queued for transmission. The implementation of the mixed message in method 2 [9] is similar to method 1 to some extent. The main difference is that the periodic transmission can be delayed until the expiry of the $MDT$ in method 2. Whereas in method 1, the periodic transmission is not delayed, in fact, the Event Timer is restarted with every sporadic transmission. The $MDT$ timer is started with every periodic or sporadic transmission of the mixed message. Hence, the worst-case periodicity of the mixed message in methods 1 and 2 can never be higher than the Inhibit Timer and the $MDT$ respectively. Therefore, the existing analyses hold intact. However, the periodic transmission is independent of the sporadic transmission in the third method [10]. The periodic timer is not reset with every sporadic transmission. The mixed message can

be queued for transmission even if the $MUT$ is not expired. The worst-case periodicity of the mixed message is neither bounded by the period nor by the $MUT$. Therefore, the existing analyses cannot be applied to the mixed messages in the third implementation method. Further, there is no free tool that is able to analyze mixed messages that are implemented using the third method. Our main goal is to develop a free tool that analyzes periodic, sporadic, and as well as mixed messages in CAN.

## 8.3 Queueing Policies and Buffer Limitations in the CAN Controllers

The different types of queueing polices implemented by the CAN device drivers and communications stacks, internal organization, and hardware limitations in the CAN controllers can have significant impact on the timing behavior of CAN messages. In this section, we discuss various queueing policies and buffer limitations in the CAN controllers.

### 8.3.1 Common Queueing Policies used in CAN Controllers

The most common queueing policies in the *nodes* connected to the CAN network are priority-based and FIFO-based policies. It should be noted that a node or an ECU contains a CAN controller. We overload the terms node, ECU and CAN controller throughout this paper.

#### Priority-ordered Queues

CAN implements priority-based arbitration which means that each node selects the highest priority message from its transmit buffers while entering into the bus arbitrations. The highest priority message among the messages selected from each node wins the bus arbitration, i.e., the right to transmit on the bus. Thus the most natural queueing policy suited to CAN controllers is priority-based queueing.

In order to demonstrate the priority based queueing policy, consider the example of three nodes namely Node A, Node B and Node C that are connected to a single CAN network as shown in Figure 8.2. Assume that each node sends three messages over the network. Node A sends the messages $m_1$, $m_3$ and $m_5$. Node B sends the messages $m_2$, $m_4$ and $m_9$. Whereas, Node C sends the messages $m_6$, $m_7$ and $m_8$. The number in the subscript denotes the

message priority. We assume that the smaller the value of the subscript, the higher the priority. Thus $m_1$ is the highest priority message, whereas $m_9$ is the lowest priority message in the system. Assume that all messages in each node are queued for transmission. In order to simplify the example, assume that the periods of all messages are very high compared to their corresponding transmission times. We also assume that there cannot be multiple instances of a message queued for transmission at the same time.

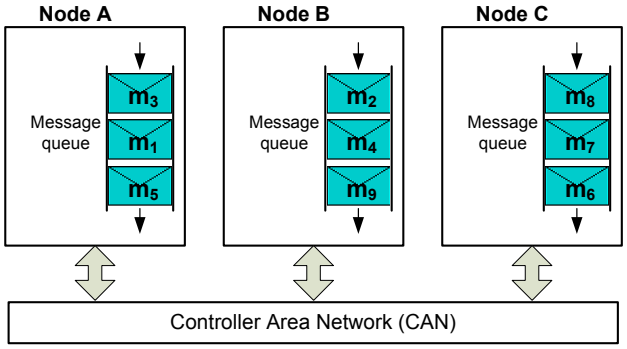


Figure 8.2: Example to demonstrate different queueing policies

Let the nodes implement priority ordered queues. Intuitively, each node will select the highest priority message from its queue to enter into the bus arbitration. In the first round, Nodes A, B, and C pick messages $m_1$, $m_2$ and $m_6$ respectively. $m_1$ wins the bus arbitration and is transmitted over the network as shown in Figure 8.3. In the second round, Nodes A, B, and C pick messages $m_3$, $m_2$ and $m_6$ respectively. This time, $m_2$ wins the bus arbitration and is transmitted over the network. Similar priority-based selection and arbitration occur during the rest of the rounds as shown in Figure 8.3.

### FIFO Queues

Due to simplicity of FIFO policy, some CAN controllers implement FIFO queues, e.g., Microchip PIC32MX, Infineon XC161CS, Renesas R32C/160 and XILINX LogiCORE IP AXI Controller [11, 12]. When the nodes implement FIFO queues, the oldest message in the transmit queue of each node competes for the bus with the oldest messages in the transmit queues in the rest of the nodes. However, the bus arbitration among these messages is done on

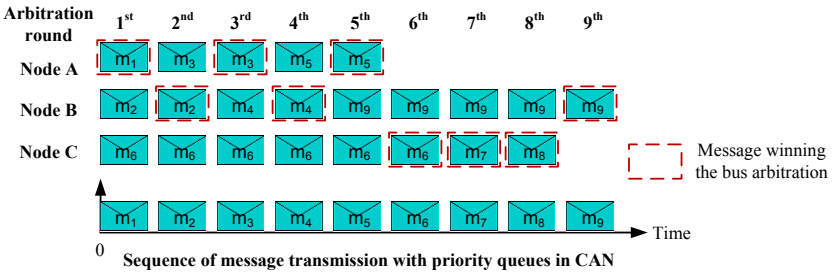


Figure 8.3: priority-based queues and CAN arbitration

priority basis. Consider again the example of the three nodes as shown in Figure 8.2. Assume that the nodes implement FIFO queues. Intuitively, each node will select the oldest message in its queue to enter into the bus arbitration. In the first round, Nodes A, B, and C pick messages $m_5$, $m_9$ and $m_6$ respectively. $m_5$ wins the bus arbitration due to its higher priority and is transmitted over the network as shown in Figure 8.4. In the second round, Nodes A, B, and C pick messages $m_1$, $m_9$ and $m_6$ respectively. This time, $m_1$ wins the bus arbitration and is transmitted over the network. Similar FIFO selection and priority-based arbitration occur during the rest of the rounds as shown in Figure 8.4.

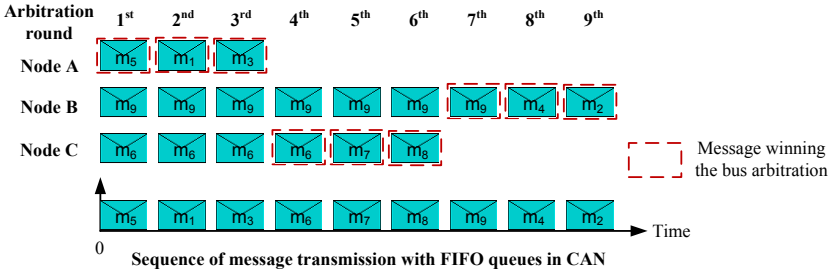


Figure 8.4: FIFO-based queues and CAN arbitration

When FIFO queues are used, the priorities of messages are often not respected in the transmit queue within a node, e.g., the lower priority message $m_5$ is transmitted before the highest priority message $m_1$ as shown in Figure 8.4. Moreover, priority inversions can occur due to which higher priority messages may have very large response times. This becomes evident by comparing

the response time of $m_2$ in the systems with priority and FIFO queues as shown in Figures 8.3 and 8.4 respectively.

### 8.3.2 Buffer Limitations in the CAN Controllers

When there are fewer number of transmit buffers in the CAN controller compared to the number of messages sent by the ECU, the messages may be subjected to extra delay and jitter due to priority inversion. Examples of the CAN controllers that implement less than three transmit buffers are 8xC592, SJA1000 and 82C200 by Philips [11, 13, 14]. If a CAN controller has less than three transmit buffers and does not support transmission abort requests as in the case of Philips 82C200, a higher priority message released in the same controller may suffer from priority inversion [13, 15, 16]. That is, if all buffers in the CAN controller are occupied by lower priority messages, a higher priority message released in the same controller has to wait for one of the lower priority messages to transmit, thereby, vacating a space in the transmit buffer. During this waiting time, priority inversion occurs that adds an additional delay to the response time of the higher priority message.

The priority inversion can occur even if the controllers support transmission abort requests. Consider the case of two transmit buffers in every CAN controller. If a higher priority message becomes ready when both transmit buffers are occupied by the lower priority messages, the lowest priority message in the transmit buffer (that is not under transmission) is swapped with the higher priority message from the message queue. During the swapping process, it may be possible that the lower priority message from the second buffer finishes its transmission and the next arbitration period starts. At this point, both buffers may be empty while any other lower priority message from another node wins the arbitration and starts to transmit. This causes priority inversion for the higher priority message that is being swapped.

In the remaining part of this subsection, we consider the CAN controllers to implement limited number (at least three) of transmit buffers. First we consider the case where the CAN controllers support transmission abort requests, e.g., Atmel AT89C51CC03/AT90CAN32/64 and Microchip MPC2515 [11]. Second we consider the case in which the CAN controllers implement non-abortable transmit buffers, e.g., Philips 82C200 [13, 15, 16].

### Additional Delay and Jitter due to Priority Inversion in the Case of Abortable Transmit Buffers

If the CAN controller supports transmission abort requests (and implements at least 3 transmit buffers) then the lowest priority message in the transmit buffer that is not undergoing transmission is swapped with the higher priority message from the message queue. During the swapping process, a lower priority message from the transmit buffer in any other controller may win the bus arbitration and contribute an extra delay to the response time of the higher priority message. The copying delay and the extra blocking delay during the swapping process should be taken into account while calculating the response time of the higher priority message.

In order to demonstrate the additional delay due to priority inversion when CAN controllers support transmission abort requests, consider the example of transmission of a message set shown in Figure 8.5. Assume there are three nodes $CC_c$, $CC_j$ and $CC_k$ in the system and each node has three transmit buffers. $m_1$ is the highest priority message in the node $CC_c$ as well as in the system. When $m_1$ becomes ready for transmission in the message queue, a lower priority message $m_6$ belonging to node $CC_k$ is already under transmission. $m_6$ cannot be preempted because CAN uses fixed priority non-preemptive scheduling. This represents the blocking delay for $m_1$. At this point in time, all transmit buffers in $CC_c$ are occupied by the lower priority messages (say $m_3$, $m_4$ and $m_5$). The device drivers signal an abort request for the lowest priority message in $K_c$ (transmit buffers in $CC_c$) that is not under transmission.

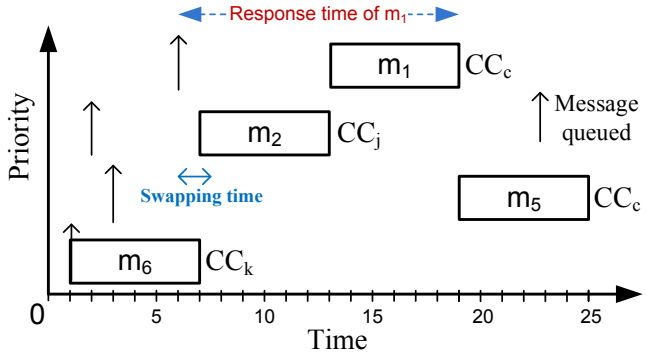


Figure 8.5: Demonstration of priority inversion in the case of abortable transmit buffers

Hence, $m_5$ is aborted and copied from the transmit buffer to the message queue, whereas $m_1$ is moved to the vacated transmit buffer. The time required to do this swapping is identified as *swapping time* in Figure 8.5. During the swapping time a series of events may occur: $m_6$ finishes its transmission, new arbitration round starts, another message $m_2$ belonging to node $CC_j$ and having priority lower than $m_1$ wins the arbitration and starts its transmission. Thus $m_1$ has to wait in the transmit buffer until $m_2$ finishes its transmission. This results in the priority inversion for $m_1$ and adds an extra delay to its response time. In [12], Khan et al. pointed out that this extra delay of the higher priority message appears as its additional jitter to the lower priority messages, e.g., $m_5$ in Figure 8.5.

**Discussion on Message Copy Time and Delay**

If the message copy time is smaller than or equal to the inter-frame space (i.e., time to transmit 3 bits on CAN bus), a lower priority message in the transmit buffer (that is not under transmission) can be swapped with a higher priority message in the message queue before the transmission of the next frame on the CAN bus [1]. Hence, there will be no priority inversion. This means that the message copy time must be, at least, $4*\tau_{bit}$ for the priority inversion to occur. Where $\tau_{bit}$ is the time required to transmit a single bit on CAN. For example, it is equal to 1 microsecond for the CAN bus speed of 1 Mbit/s. In Legacy systems, there may be slow controllers, i.e., the speed of the controllers can be slower than the maximum operating speed of the CAN bus (1 Mbit/s). Since the amount of data transmitted in a CAN message rages from 0 to 8 bytes, the transmission time of a message also varies accordingly. According to [17], the transmission time of a CAN message with standard frame format ranges from $55*\tau_{bit}$ to $135*\tau_{bit}$ for the amount of data contained in the message that ranges from 0 to 8 bytes respectively. Let us assume the message copy time to be equal to $4*\tau_{bit}$. Intuitively, the message copy time can range from 7.3% to 3% of transmission time of a message with 0 to 8 bytes of data respectively. Due to slow controllers that may be found in legacy systems, the message copy time can be greater than $4*\tau_{bit}$. Hence, the message copy time can be higher than 7.3% of its transmission time.

### Additional Delay and Jitter due to Priority Inversion in the Case of Non-abortable Transmit Buffers

When CAN controllers do not support transmission abort requests, a higher priority message may suffer from priority inversion and this, in turn, may add extra delay to its response time [13]. Consider an example of three controllers $CC_c$, $CC_j$, $CC_k$ connected to a single CAN network in Figure 8.6. Let $m_1$, belonging to $CC_c$, be the highest priority message in the system. Assume that when $m_1$ is ready to be queued, all transmit buffers in $CC_c$ are occupied by lower priority messages which cannot be aborted because the controllers implement non-abortable transmit buffers. In addition, $m_1$ can be blocked by any lower priority message because the lower priority message already started its transmission. In this example $m_1$ is blocked by $m_5$ that belongs to node $CC_k$. Since all transmit buffers in $CC_c$ are full, $m_1$ has to wait in the message queue until one of the messages in the transmit buffers of node $CC_c$ is transmitted.

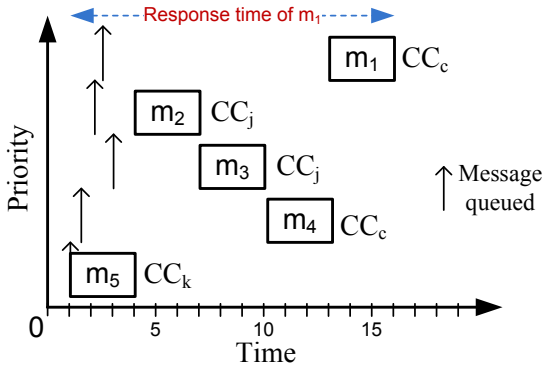


Figure 8.6: Demonstration of priority inversion in the case of non-abortable transmit buffers

Let $m_4$ be the highest priority message in the transmit buffers of node $CC_c$. $m_4$ can be interfered by higher priority messages ($m_2$ and $m_3$) belonging to other nodes. Hence, it can be seen that priority inversion for $m_1$ takes place because $m_1$ cannot start its transmission before $m_4$ finishes its transmission, while $m_4$ has to wait until messages $m_2$ and $m_3$ are transmitted. This adds an additional delay to the worst-case response time of $m_1$. In this example, this additional delay is the sum of the worst-case transmission times of $m_2$, $m_3$ and $m_4$. This additional delay appears as additional jitter of $m_1$ as seen by the

lower priority messages.

## 8.4 Related Works

### 8.4.1 Related Analyses

Tindell et al. [16] developed the schedulability analysis for CAN. It has been implemented in the analysis tools that are used in the automotive industry, e.g., Volcano Network Architect (VNA) [18]. Davis et al. [17] refuted, revisited and revised the seminal analysis of [16]. The revised analysis is implemented in the existing industrial tool suite Rubus-ICE [19, 20]. These analyses assume that each node selects the highest priority message, that is ready for transmission, from its transmit buffers when entering into the bus arbitration. It is noted in [11, 12, 13, 14, 15, 21, 22, 23, 24] that this assumption may become invalid in some cases due to various practical limitations such as controllers implementing FIFO and work-conserving queues, limited number of transmit buffers, copying delays in transmit buffers, transmit buffers supporting abort requests and protocol stack prohibiting transmission abort requests in some configurations as in the case of AUTOSAR [25].

In [11, 14, 24], Davis et al. extended the analysis of CAN with FIFO and work-conserving queues while supporting arbitrary deadlines of messages. In [22], Meschi et al. proved the priority inversion due to limited buffers can be avoided if the controller implements at least three transmit buffers. However, the analysis in [22] does not account the overhead of the copying delay. Khan et al. [12] integrated this extra delay with the analysis in [16, 17] for the case of abortable transmit buffers. In the case of CAN controllers implementing non-abortable transmit requests, RTA for CAN is extended in [13, 15]. But, none of the analysis discussed above supports messages that are scheduled with offsets. The worst-case RTA for CAN messages with offsets has been developed in several works [26, 27, 28, 29, 30].

However, all these analyses assume that the messages are queued for transmission either periodically or sporadically. They do not support mixed messages that are partly periodic and partly sporadic. Mubeen et al. [31] extended the seminal and revised analyses [16, 17] to support mixed messages in CAN where nodes implement priority queues. Mubeen et al. [32] further extended their analysis to support mixed messages in CAN where some nodes implement priority queues while others implement FIFO queues. In [33] and [34] we extended the analysis for mixed messages in CAN where the controllers

implement abortable and non-abortable transmit buffers respectively. Mubeen et al. also extended the existing analysis for CAN to support periodic, sporadic and mixed messages that are scheduled with offsets [35, 36].

### 8.4.2 Related Tools

VNA [18] is a communication design tool that supports RTA for CAN. It implements RTA of CAN developed by Tindell et al. [16].

Vector[2] is a tools provider for the development of networked electronic systems. CANalyzer [37] supports the simulation, analysis and data logging for the systems that use CAN for network communication. CANoe [38] is a tool for the simulation of functional and extra-functional (e.g., timing) behavior of ECU networks. Network Designer CAN is another tool by Vector that is used to design the architecture and perform timing analysis of CAN.

SymTA/S [39] is a tool by Symtavision for model-based timing analysis and optimization. Among other analyses, it supports statistical, and worst- and best-case timing analysis for CAN.

RTaW-Sim [40] is a tool for the simulation and performance evaluation of the CAN network.

The Rubus-ICE is a commercial tool suite developed by Arcticus Systems[3] in close collaboration with Mälardalen University Sweden. It supports model- and component-based development of real-time embedded systems[41, 42]. Among other analyses, it supports RTA of CAN [16, 17] and RTA of CAN for mixed messages[31].

To the best of our knowledge, there is no freely-available tool that implements RTA of CAN for mixed messages. The main purpose of MPS-CAN analyzer is to support RTA of periodic, sporadic and mixed messages in CAN while taking into account different queueing policies and buffer limitations in the CAN controllers and device drivers.

### 8.4.3 Extended Version

This paper extends our previous work [43] where we discussed the implementation of RTA for periodic, sporadic and mixed messages in CAN without considering hardware and software limitations in the CAN controllers and device drivers. In the extended version of the paper, we discuss the integration of these limitations with the response-time analysis for CAN and implementation

[2] http://www.vector.com
[3] http://www.arcticus-systems.com

of the analyses in the tool. Moreover, we conduct a detailed case study from the automotive domain. We also evaluate the implemented analysis.

## 8.5 Implemented Analyses, Layout and Usage of the Tool

### 8.5.1 Analyses Implemented in the MPS-CAN Analyzer

The analyses that we implemented in the MPS-CAN analyzer consist of RTA for CAN and its several extensions as shown in Figure 8.7. The figure also shows the relationship among the implemented analyses. We denote each extension of the RTA for CAN by the term "analysis profile".

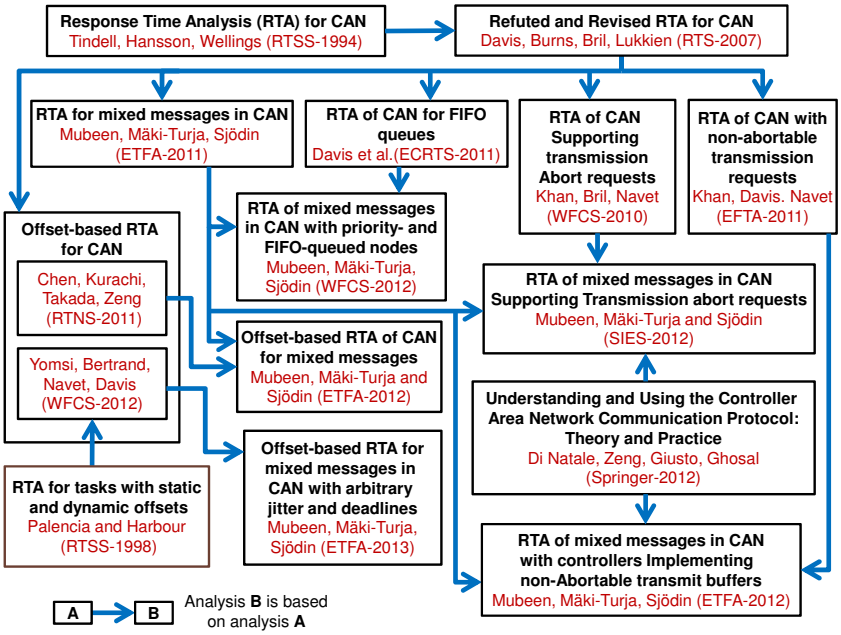


Figure 8.7: Graphical representation of the Response Time Analysis (RTA) and its extensions implemented in the MPS-CAN analyzer

### 8.5.2 Implementation and Distribution

We developed an algorithm that integrates the analysis profiles that are shown in Figure 8.7. It also shows the high-level implementation of the analyses in the MPS-CAN analyzer as depicted in Algorithm 2. The MPS-CAN analyzer is implemented in the C language. The graphical user interface of the tool is developed using the Windows Application Programming Interface (WinAPI). Each analysis profile supported by the tool is implemented as a separate C file which is accessed using function calls in the main file. The Figure 8.8 shows the screen shot of the code where a number of functions corresponding to different analyses are shown. A new analysis can be easily added to the MPS-CAN analyzer by adding a similar function and corresponding source files (.c and .h) provided the new analysis complies with the input and output interfaces shown in the structures in Figure 8.9. Hence, the tool supports a simple and easy mechanism for further extensions and implementation of other related analyses in the future. The tool, user manual, and test cases can be downloaded at https://github.com/saadmubeen/MPS-CAN.

```
751
752         analysis_type.analysis_property = PRIORITY;
753         analysis_type.offset = OFFSET_UNAWARE;
754
755         Compute_Rm_of_all_MSGs_Mixed_Prio();
756
757         Compute_Rm_of_ ll_MSGs_Mixed_Abort();
758
759         Compute_Rm_of_all_MSGs_Mixed_NonAbort();
760
761         Compute_Rm_of_all_MSGs_Mixed_Prio_Offset();
762
763         Compute_Rm_of_all_MSGs_Mixed_FIFO();
```

Figure 8.8: Screen shot from the code: functions corresponding to different analyses

### 8.5.3 Tool Layout, Inputs and Outputs

The Layout of the MPS-CAN analyzer is shown in Figure 8.10. There is a main window denoted by “MPS-CAN Analyzer” which serves as the user interface. The input section of the tool consists of the list boxes (“Message List”, “Node

**Algorithm 2** Algorithm for high-level implementation of the analyses

```
1: begin
2: RT_Prev ← 0                          ▷ Initialize all Response Times (RTs) to zero
3: READ_INPUT ()                        ▷ Bus speed, ECUs, and messages input
4: procedure CALCULATE_MESSAGE_RESPONSE_TIME ()
5:     if message_under_analysis ∈ ECU_with_prrioty_queue then
6:         if buffer_type == no_limitaton then
7:             RTA_OF_CAN_WITH_PRIORITY_QUEUES ()
8:         else if buffer_type == abort then
9:             RTA_OF_CAN_ABORTABLE_TRASNSMIT_BUFFERS ()
10:        else if buffer_type == non_abort then
11:            RTA_OF_CAN_NON-ABORTABLE_TRASNSMIT_BUFFERS ()
12:        end if
13:    else
14:        RTA_OF_CAN_WITH_FIFO_QUEUES ()
15:    end if
16: end procedure
17: for all Messages_in_the_system do
18:    Repeat ← TRUE
19:    while Repeat = TRUE do
20:        if messages_are_scheduled_with_offsets == FALSE then
21:            CALCULATE_MESSAGE_RESPONSE_TIME ()
22:        else
23:            CALCULATE_MESSAGE_RESPONSE_TIME_WITH_OFFSETS ()
24:        end if
25:        if RT > RT_Prev then
26:            RT_Prev ← RT
27:            Repeat ← TRUE
28:        else
29:            Repeat ← FALSE
30:        end if
31:        if RT ≥ Deadline then
32:            Repeat ← FALSE
33:        end if
34:    end while
35: end for
36: end
```

```
typedef struct ECU_TYPE
{
    long int    ID;                 // Node ID
    buf_type    buffer_type;        // abortable, non-abortable, not aplicable--Added for CAN journal analysis
    long int    Kc;                 // Size of transmission buffer
    long int    max_Pm_Kc;          // Priority of highest priority message in Kc for the respective node
    long int    nr_of_msgs;         // Total number of messages belonging to the ECU
} ECU_type;

typedef struct message
{
    long int    ID;                 // Reference to corresponding msg ID in Network Specification
    long int    FRAME_TYPE;         // Standard(11-bit) or extended(29-bit)
    long int    TRANSMISSION_TYPE;  // PERIODIC, EVENT(e.g. All signals in frame are of event type), MIXED
    long int    Tm;                 // Message Transmission Period
    long int    MIN_UPDATE_TIME;    // Minimum Update Time for Event and Mixed Transmission of the msg,
    long int    Dm;                 // Message deadline i.e. from queuing the msg to delivery to the receiving CAN controller
    long int    DLC;                // Data Length Code: Number of data bytes in a CAN Data Frame (0-8 bytes)
    long int    Jm;                 // Release Jitter of a message inherited as WCRT from the task producing it
    long int    Om;                 // Offset
    long int    CC;                 // Sender Node
    long int    Cm;                 // Transmission time
    int         Cm_computation;     // Calculated Cm
    long int    ECU_ID;             // The node to which this msg belongs
    long int    CTm;                // Copy Time
    CTm_input   CTm_in;             // How CTm has been provided: DEFAULT (more than Jbits, decide later), PERCENTAGE, USER_DEFINED
    long int    Jm_hat_A;           // Total jitter
    long int    Bm_hat_A;           // Additional blocking
    long int    AJm_A;              // Addition jitter
    long int    Rm_P;               // Worst Case Response-Time of a Periodic copy of MIXED message
    long int    Rm_S;               // Worst Case Response-Time of a Sporadic copy of MIXED message
    long int    Rm;                 // Worst Case Response-Time of a message
    MSG_trace_type  MSG_trace;      // Contains the linking information of the message, i.e., who is the sender and who are the receivers

} MSGtype;
```

Figure 8.9: Screen shot from the code: structures for inputs and outputs

List", "Network Speed" and "Number of Nodes") and buttons. Whereas, the output section of the tool comprises of the list boxes namely "Output", "Network Utilization", and "Errors and Warnings".

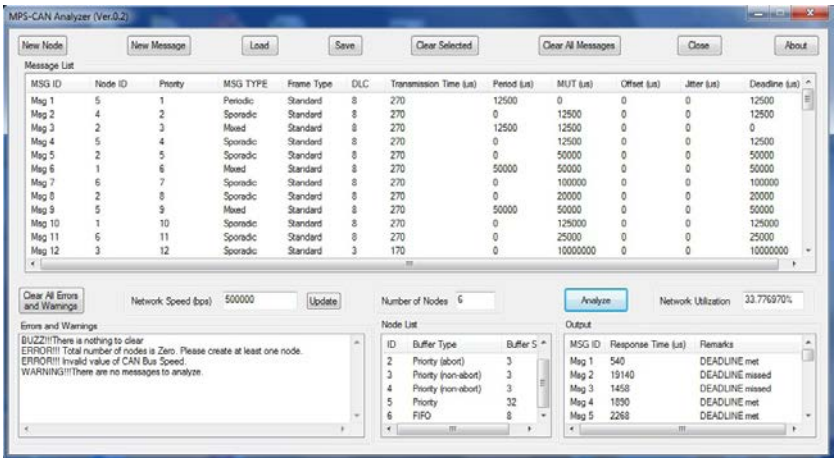


Figure 8.10: MPS-CAN analyzer layout, inputs and outputs

When the "New Node" button is clicked on the main window, a new window namely "New Node" opens up as shown in Figure 8.11. This window is used to create a new node. In this window the user can specify the node ID and the number of transmission buffers in the node. This window also allows implicit selection of the analysis profiles. If the selected type of transmit buffers is "Priority (no buffer limitations)", the node is assumed to implement priority-based queueing policy. Furthermore, the node contains very high (but finite) number of transmit buffers compared to the number of messages that are sent by this node. In this case, the RTA for mixed, periodic, and sporadic messages without any buffer limitations is used to analyze all messages that are sent by this node.

If the selected type of transmit buffers is "Priority (abortable buffer)" or "Priority (non-abortable buffer)", the node contains limited (at least three) number of transmit buffers which are of abortable or non-abortable type respectively. In both of these cases, the node is assumed to implement priority-based queueing policy. In these two cases, the RTA for mixed, periodic, and sporadic messages supporting abortable or non-abortable transmit buffers is used to analyze all messages that are sent by this node respectively. Similarly, if the selected type of transmit buffers is "FIFO", the node is assumed to implement FIFO-based queueing policy. In this case, the RTA for mixed, periodic, and sporadic messages in CAN with FIFO queues is used to analyze all messages that are sent by this node.

When the "New Message" button is clicked on the main window, a new window namely "New Message" pops up as shown in Figure 8.12. This window is used to create a new message. In this window, message attributes are provided as input. For a mixed message, both period and minimum update time are specified. Whereas for a periodic or sporadic message, only period or minimum update time is specified respectively. The transmission type of a message can be selected from periodic, sporadic, or mixed. There are two options for specifying transmission type of a message. First option is based on specifying Data Length Code (DLC), i.e., the number of data bytes present in the CAN message. The second option allows to specify user-defined transmission time. This option may be used for analyzing simplified test cases that are more suitable for research-oriented work. There are several options to select and specify "Message Copy Time" which is the time required to copy a message from the transmit buffer to the message queue or vice versa. If the message offset is specified, then the messages are analyzed using the offset-based RTA for mixed, periodic, and sporadic messages in CAN.

In the main window, the network speed in bits per second (bps) can be spec-

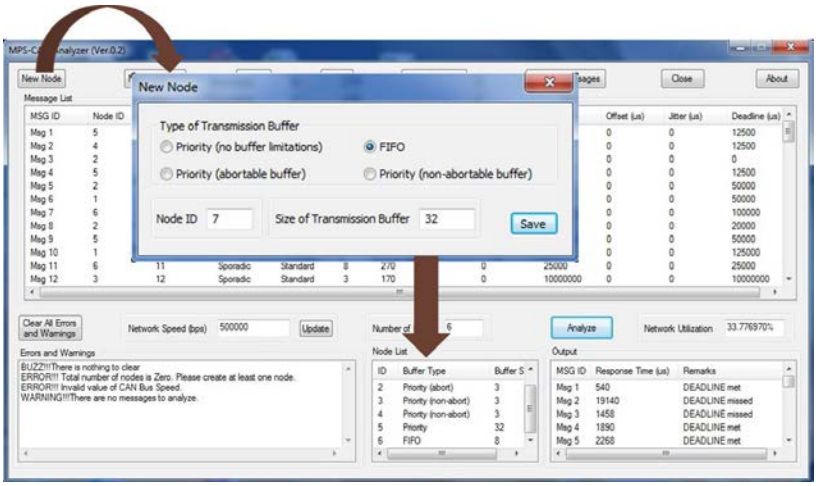


Figure 8.11: Creating a new node and implicitly selecting the RTA in the MPS-CAN analyzer

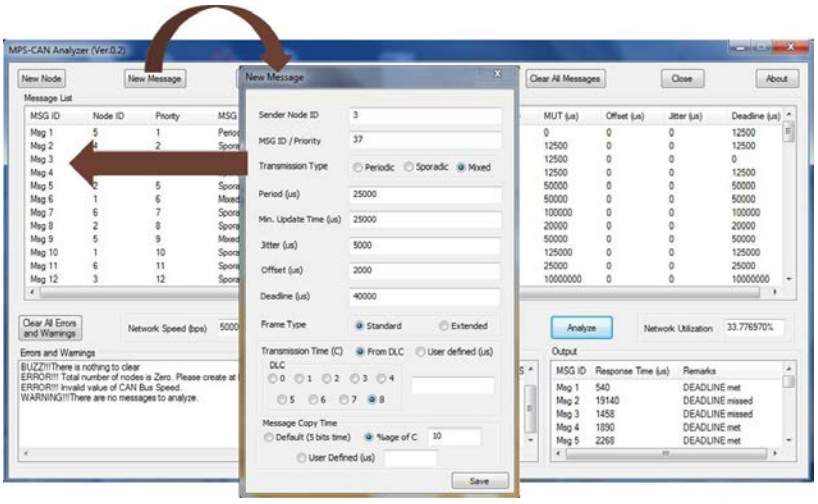


Figure 8.12: Creating a new message in the MPS-CAN analyzer

ified. Moreover, there are buttons provided to clear, save and load messages. Any message set can be analyzed by clicking the "Analyze" button. If errors and warnings occur during the run of the analyzer, they are displayed in the "Errors and Warnings" list box. Figure 8.10 shows some errors and warnings that may occur when the analyzer is run. The "Output" list box displays the calculated response times of the messages. It also displays whether a message meets its deadline or not (provided the deadline is specified by the user). The percentage network utilization is also calculated and displayed in the "Network Utilization" list box.

## 8.6 Case study and Evaluation

In order to show the usability of the MPS-CAN analyzer, we conduct an automotive-application case study. Basically, we adapt the case study of the experimental vehicle that is discussed and analyzed in [21].

### 8.6.1 Experimental Setup

The system model in the original experimental vehicle consists of 6 identical ECUs (identical in terms of buffer limitations) that are connected to a single CAN network. There are 81 periodic CAN messages in the system. We adapt this system in such a way that it becomes heterogeneous in terms of different queueing policies and buffer limitations in the ECUs. However, the number of ECUs and messages remains unchanged. That is, the modified experimental vehicle contains six ECUs out of which two use priority-based queueing policy and each of them implements 3 abortable transmit buffers; two use priority-based queueing policy and each of them implements 3 non-abortable transmit buffers; one implements FIFO queue with 8 buffers; and the remaining ECU uses priority-based queueing policy and has no buffer limitations which means that it implements very large but finite number of transmit buffers (32 buffers). The 81 messages are equally assigned different transmission types. This means, there are 27 periodic, 27 sporadic, and 27 mixed messages in the system.

All the attributes of these messages are tabulated in Figure 8.13. The attributes of each message are identified as follows. The priority, sender ECU ID, type of transmit buffers implemented by the sender ECU, transmission type, number of data bytes in the message, transmission period, minimum update time, deadline, and calculated worst-case response time are represented by

| Prio | ECU ID | ECU Type | ξ | DLC(byte) | T (us) | MUT (us) | D (us) | R (us) | Prio | ECU ID | ECU Type | ξ | DLC(byte) | T (us) | MUT (us) | D (us) | R (us) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 5 | Prio-No-Limit | P | 8 | 12500 | 0 | 12500 | 540 | 42 | 2 | Abort | P | 8 | 100000 | 0 | 100000 | 16748 |
| 2 | 4 | Non-Abort | S | 8 | 0 | 12500 | 25000 | 19140 | 43 | 4 | Non-Abort | S | 8 | 0 | 100000 | 100000 | 22110 |
| 3 | 2 | Abort | M | 8 | 12500 | 12500 | 12500 | 1458 | 44 | 6 | FIFO | P | 8 | 100000 | 0 | 100000 | 32610 |
| 4 | 5 | Prio-No-Limit | S | 8 | 0 | 12500 | 12500 | 1890 | 45 | 5 | Prio-No-Limit | S | 8 | 0 | 50000 | 50000 | 17450 |
| 5 | 2 | Abort | S | 8 | 0 | 50000 | 50000 | 2268 | 46 | 4 | Non-Abort | P | 8 | 50000 | 0 | 50000 | 22380 |
| 6 | 1 | Abort | M | 8 | 50000 | 50000 | 50000 | 2538 | 47 | 1 | Abort | S | 8 | 0 | 50000 | 50000 | 18368 |
| 7 | 6 | FIFO | S | 8 | 0 | 100000 | 100000 | 32610 | 48 | 4 | Non-Abort | M | 8 | 50000 | 50000 | 50000 | 22650 |
| 8 | 2 | Abort | S | 8 | 0 | 20000 | 20000 | 3348 | 49 | 1 | Abort | S | 8 | 0 | 1000000 | 1000000 | 19448 |
| 9 | 5 | Prio-No-Limit | M | 8 | 50000 | 50000 | 50000 | 3510 | 50 | 3 | Non-Abort | P | 8 | 1000000 | 0 | 1000000 | 29670 |
| 10 | 1 | Abort | S | 8 | 0 | 125000 | 125000 | 4158 | 51 | 4 | Non-Abort | S | 8 | 0 | 1000000 | 1000000 | 23190 |
| 11 | 6 | FIFO | S | 8 | 0 | 25000 | 35000 | 32610 | 52 | 6 | FIFO | P | 8 | 1000000 | 0 | 1000000 | 32610 |
| 12 | 3 | Non-Abort | S | 3 | 0 | 10000000 | 10000000 | 26700 | 53 | 3 | Non-Abort | M | 8 | 128000 | 128000 | 128000 | 29940 |
| 13 | 6 | FIFO | M | 8 | 100000 | 100000 | 100000 | 32730 | 54 | 2 | Abort | S | 8 | 0 | 128000 | 128000 | 22418 |
| 14 | 4 | Non-Abort | P | 8 | 100000 | 0 | 100000 | 20760 | 55 | 1 | Abort | P | 8 | 128000 | 0 | 128000 | 22688 |
| 15 | 6 | FIFO | M | 8 | 100000 | 100000 | 100000 | 32730 | 56 | 4 | Non-Abort | M | 8 | 1000000 | 1000000 | 1000000 | 23460 |
| 16 | 6 | FIFO | M | 8 | 100000 | 100000 | 100000 | 32730 | 57 | 4 | Non-Abort | S | 8 | 0 | 250000 | 250000 | 24000 |
| 17 | 5 | Prio-No-Limit | S | 8 | 0 | 100000 | 100000 | 7190 | 58 | 3 | Non-Abort | M | 3 | 250000 | 250000 | 250000 | 30380 |
| 18 | 5 | Prio-No-Limit | P | 8 | 1000000 | 0 | 1000000 | 7460 | 59 | 4 | Non-Abort | M | 8 | 500000 | 500000 | 500000 | 24000 |
| 19 | 4 | Non-Abort | S | 8 | 0 | 1000000 | 1000000 | 21030 | 60 | 2 | Abort | M | 8 | 500000 | 500000 | 500000 | 24648 |
| 20 | 1 | Abort | P | 8 | 1000000 | 0 | 1000000 | 8108 | 61 | 5 | Prio-No-Limit | M | 7 | 500000 | 500000 | 500000 | 25060 |
| 21 | 5 | Prio-No-Limit | P | 8 | 1000000 | 0 | 1000000 | 8270 | 62 | 1 | Abort | M | 8 | 500000 | 500000 | 500000 | 26714 |
| 22 | 1 | Abort | M | 8 | 500000 | 500000 | 500000 | 8648 | 63 | 1 | Abort | S | 2 | 0 | 500000 | 500000 | 27110 |
| 23 | 1 | Abort | P | 8 | 500000 | 0 | 500000 | 9188 | 64 | 1 | Abort | M | 8 | 1000000 | 1000000 | 1000000 | 27404 |
| 24 | 3 | Non-Abort | S | 8 | 0 | 500000 | 500000 | 26970 | 65 | 2 | Abort | P | 8 | 1000000 | 0 | 1000000 | 28808 |
| 25 | 4 | Non-Abort | P | 8 | 500000 | 0 | 500000 | 21300 | 66 | 2 | Abort | M | 8 | 1000000 | 1000000 | 1000000 | 29078 |
| 26 | 2 | Abort | P | 8 | 100000 | 0 | 100000 | 9998 | 67 | 2 | Abort | P | 8 | 1000000 | 0 | 1000000 | 29564 |
| 27 | 3 | Non-Abort | S | 8 | 0 | 100000 | 100000 | 27240 | 68 | 3 | Non-Abort | P | 8 | 1000000 | 0 | 1000000 | 31090 |
| 28 | 1 | Abort | P | 8 | 100000 | 0 | 100000 | 10538 | 69 | 6 | FIFO | P | 6 | 1000000 | 0 | 1000000 | 32610 |
| 29 | 3 | Non-Abort | S | 8 | 0 | 1000000 | 1000000 | 27510 | 70 | 5 | Prio-No-Limit | S | 8 | 0 | 2000000 | 2000000 | 30280 |
| 30 | 5 | Prio-No-Limit | M | 8 | 1000000 | 1000000 | 1000000 | 10970 | 71 | 6 | FIFO | S | 8 | 0 | 2000000 | 2000000 | 32610 |
| 31 | 5 | Prio-No-Limit | S | 8 | 0 | 1000000 | 1000000 | 11510 | 72 | 3 | Non-Abort | P | 8 | 2000000 | 0 | 2000000 | 31090 |
| 32 | 2 | Abort | M | 8 | 20000 | 20000 | 30000 | 11888 | 73 | 3 | Non-Abort | M | 8 | 2000000 | 2000000 | 2000000 | 31360 |
| 33 | 1 | Abort | S | 8 | 0 | 50000 | 50000 | 12428 | 74 | 4 | Non-Abort | M | 8 | 2000000 | 2000000 | 2000000 | 32170 |
| 34 | 5 | Prio-No-Limit | M | 8 | 500000 | 500000 | 500000 | 12590 | 75 | 2 | Abort | S | 8 | 0 | 2000000 | 2000000 | 32764 |
| 35 | 5 | Prio-No-Limit | P | 8 | 20000 | 0 | 50000 | 14210 | 76 | 6 | FIFO | P | 8 | 2000000 | 0 | 2000000 | 32610 |
| 36 | 4 | Non-Abort | P | 8 | 500000 | 0 | 500000 | 21570 | 77 | 2 | Abort | M | 8 | 2000000 | 2000000 | 2000000 | 33304 |
| 37 | 5 | Prio-No-Limit | P | 8 | 20000 | 0 | 50000 | 14750 | 78 | 6 | FIFO | M | 2 | 2000000 | 2000000 | 2000000 | 32610 |
| 38 | 6 | FIFO | S | 8 | 0 | 200000 | 200000 | 32610 | 79 | 4 | Non-Abort | M | 1 | 50000 | 50000 | 100000 | 33830 |
| 39 | 3 | Non-Abort | P | 8 | 20000 | 0 | 50000 | 27780 | 80 | 6 | FIFO | M | 2 | 1000000 | 1000000 | 1000000 | 32610 |
| 40 | 1 | Abort | P | 8 | 200000 | 0 | 200000 | 16208 | 81 | 3 | Non-Abort | M | 2 | 2000000 | 2000000 | 2000000 | 33410 |
| 41 | 3 | Non-Abort | P | 8 | 1000000 | 0 | 1000000 | 28590 | | | | | | | | | |

Figure 8.13: Attributes and calculated response times of periodic, sporadic and mixed messages in the automotive case study

$Prio$, $ECU\_ID$, $ECU\_Type$, $\xi$, $DLC$, $T$, $MUT$, $D$, and $R$ respectively. We assume, the smaller the value of the $Prio$ parameter of a message, the higher its priority. Thus, the message with priority 1 is the highest priority message, whereas the message with priority 81 is the lowest priority message in the system under analysis. We assume that the copy time of each message is more than the time required to transmit 4 bits on the CAN bus. For simplicity, the copy time of each message is selected to be 10% of its transmission time. All timing parameters are in microseconds. The selected speed for CAN is 500 Kbit/s.

The MPS-CAN analyzer treats each message differently depending upon its transmission type; and the type of queueing policy and buffer limitations in the sender ECU. The worst-case response times of all messages calculated by the MPS-CAN analyzer are listed in Figure 8.13. The network utilization calculated by the MPS-CAN analyzer for this message set is equal to 33.776970%. The tool takes less than 2 seconds to analyze the case study on a laptop with

dual core 2.4 GHz processor, 2 GB RAM and Windows (OS). By comparing the calculated response time with the corresponding deadline of each message in the table, it is obvious that all messages meet their deadlines. Hence, the heterogeneous system is schedulable.

### 8.6.2 Comparison of Various Response-time Analyses

In order to compare the response times calculated from different analyses in the MPS-CAN analyzer, we perform four more tests on four different sets of ECUs. There are identical ECUs in each set. The same message set is analyzed in all tests. In the first test, each ECU uses priority-based queueing policy and implements large but finite number of transmit buffers (32 in this case). In this test we use the analysis for mixed, periodic, and sporadic messages in CAN with priority queues and no buffer limitations. In the second test, each ECU uses priority-based queueing policy and implements 3 abortable transmit buffers. In this test, we use the analysis for mixed, periodic, and sporadic messages in CAN with abortable transmit buffers. In the third test, each ECU uses priority-based queueing policy and implements 3 transmit buffers which are of non-abortable type. In this test we use the analysis for mixed, periodic, and sporadic messages in CAN with non-abortable transmit buffers available. Whereas, in the fourth test, each ECU uses FIFO-based queueing policy and implements 8 transmit buffers. In this test, the same message set is analyzed using the analysis for mixed, periodic, and sporadic messages in CAN with FIFO queues.

The response times of all messages in these four cases along with the response times of messages in the heterogeneous system are shown by the bar graphs in Figure 8.14. The results indicate that the message response times are the best (smallest) in the first test. This is because the corresponding analysis assumes the ideal behavior of the CAN controllers, i.e., no buffer limitations, and hence, no extra delays due to priority inversion. The second best response times are obtained in the second test. The response times in this test are higher than the response times in the first test due to the copying delay and extra delay because of the priority inversion discussed in the Section 8.3.2. The third best response times are obtained in the third test. However, these response times are considerably large compared to the response times in the first and second tests. This is because of the extra delay due to priority inversion discussed in the Section 8.3.2. Due to priority inversion, some higher priority messages have larger response times compared to the lower priority messages. For example, the response time of message with priority 2 is higher than the response

time of the message with priority 10. On the average, the response times of the messages in the heterogeneous system are comparable to the response times of the messages in the third test. Finally, the response times of the messages are the worst (largest) in the fourth test. The response times in this case are significantly large compared to the first two tests because of large delays due to priority inversion within the FIFO queues as discussed in the Section 8.3.1.

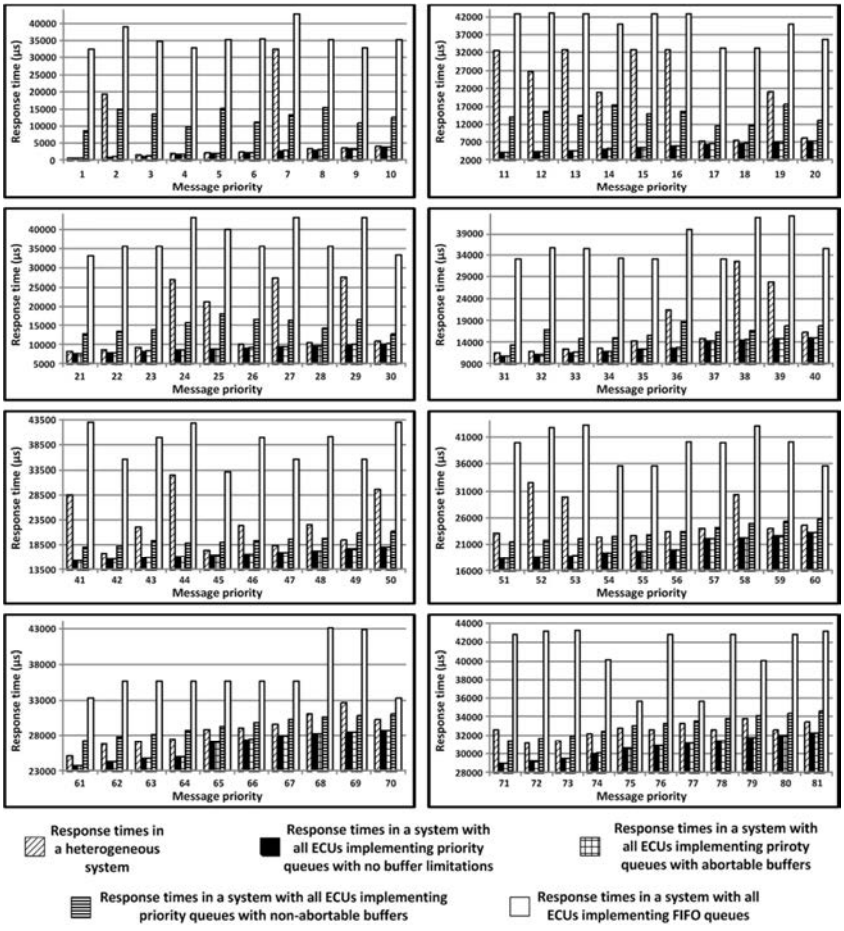


Figure 8.14: Comparison of message response-times with respect to different types of buffer limitations in the ECUs

### 8.6.3 Discussion

In order to get short response times of CAN messages, those ECUs should be selected which use priority-based queueing policy and implement much higher number of transmit buffers compared to the number of messages sent by them. However, practical systems use ECUs with limited number of transmit buffers. If ECUs with very large number of transmit buffers are not available then the ECUs with abortable transmit buffers should be preferred over the ECUs that implement non-abortable transmit buffers. Although FIFO policy is easy to implement and simple to use as compared to the priority queueing policy, the messages can have very large worst-case response times in the case of ECUs implementing FIFO queues. The ECUs which implement priority-based queueing policy should be preferred over the ECUs which implement FIFO queues especially in high utilization systems.

Moreover, it is important to use the right RTA that correctly matches the queueing policies; buffer limitation in the CAN controllers; and transmission type of messages used in the higher-level protocols. If the practical limitations and constraints are not considered in the RTA, the calculated response times can be optimistic. The MPS-CAN analyzer considers these limitations and constraints while analyzing the CAN messages. It treats each message differently based on its transmission type, and queueing policy and buffer limitations in the CAN controller of its sender ECU.

## 8.7 Conclusion

We introduced a new tool MPS-CAN analyzer to support Response Time Analysis (RTA) of periodic, sporadic and mixed messages in the Controller Area Network (CAN). The existing RTA tools for CAN analyze only periodic and sporadic messages. They do not support the analysis of mixed messages which are partly periodic and partly sporadic. These messages are implemented by several higher-level protocols for CAN that are used in the automotive industry today.

The MPS-CAN analyzer implements various extensions of the RTA for CAN while taking into account mixed messages, messages scheduled with offsets, messages with arbitrary jitter and deadlines, various queueing policies (e.g., priority- or FIFO-based), and limitations of transmit buffers in the CAN controllers (e.g., abortable or non-abortable). With the implementation of these analyses, the MPS-CAN analyzer is able to analyze network communications

in heterogeneous systems which may consist of different types of ECU's supplied by different Tier 1 suppliers.

We also showed the usability of the MPS-CAN analyzer by conducting the case study of a heterogeneous automotive application where ECUs use different queueing policies and have different buffer limitations, i.e., some have a very large number of transmit buffers, whereas, some have limited number of transmit buffers with some supporting transmission abort requests while others don't. In this application, we considered a large message set consisting of periodic, sporadic, and mixed messages. By evaluating the case study, we showed that it is important to use the RTA that matches the actual limitations and constraints in the hardware, device drivers and protocol stack. Otherwise, the calculated response times can be optimistic.

The structural organization of the MPS-CAN analyzer provides ease for further extensions and implementations of other related analyses. Since, this tool is freely available, we believe, it may prove helpful in the research-oriented projects that require the analysis of CAN-based systems.

## Acknowledgement

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

# Bibliography


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

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

[3] Automotive networks. CAN in Automation (CiA), 2011. http://www.can-cia.org/index.php?id=416.

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

[5] 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.

[6] 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.

[7] M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer Journal,*, 29(5):390–395, 1986.

[8] CANopen Application Layer and Communication Profile. CiA Draft Standard 301. Ver. 4.02. Feb., 2002. http://www.can-cia.org/index.php?id=440.

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

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

[11] Rob Davis, Steffen Kollmann, Victor Pollex, and Frank Slomka. Schedulability analysis for controller area network (CAN) with FIFO queues priority queues and gateways. *Real-Time Systems*, 49(1):73–116, 2013.

[12] D.A. Khan, R.J. Bril, and N. Navet. Integrating hardware limitations in CAN schedulability analysis. In *8th IEEE International Workshop on Factory Communication Systems (WFCS), May, 2010*, pages 207 –210, May 2010.

[13] D.A. Khan, R.I. Davis, and N. Navet. Schedulability analysis of CAN with non-abortable transmission requests. In *16th IEEE Conference on Emerging Technologies Factory Automation (ETFA), Sep., 2011*, Sep. 2011.

[14] R. Davis and N. Navet. Controller area network (CAN) schedulability analysis for messages with arbitrary deadlines in FIFO and work-conserving queues. In *9th IEEE International Workshop on Factory Communication Systems (WFCS), May, 2012*, pages 33 –42, May 2012.

[15] Marco Di Natale. Evaluating message transmission times in Controller Area Networks without buffer preemption. In *8th Brazilian Workshop on Real-Time Systems, 2006*, 2006.

[16] 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.

[17] 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.

[18] Volcano Network Architect. Mentor Graphics, http://www.mentor.com/products/vnd/communication-management/vna.

[19] Rubus-ICE: Integrated component Development Environment, 2013. http://www.arcticus-systems.com.

[20] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Support for end-to-end response-time and delay analysis in the industrial tool suite: Issues, experiences and a case study. *Computer Science and Information Systems, ISSN: 1361-1384,*, 10(1), 2013.

[21] Marco Di Natale and Haibo Zeng. Practical issues with the timing analysis of the Controller Area Network. In *18th IEEE Conference on Emerging Technologies and Factory Automation (ETFA), Sep., 2013*, Sep. 2013.

[22] A. Meschi, M. Di Natale, and M. Spuri. Priority inversion at the network adapter when scheduling messages with earliest deadline techniques. In *Eighth Euromicro Workshop on Real-Time Systems, 1996*, pages 243–248, 1996.

[23] Marco Di Natale, Haibo Zeng, Paolo Giusto, Arkadeb Ghosal. *Understanding and Using the Controller Area Network Communication Protocol*. Springer, 2012.

[24] 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, July, 2011*, July 2011.

[25] Transmit cancellation in AUTOSAR Specification of CAN Driver, Release 4.0, Rev 3, Ver. 4.0. Nov., 2012. http://www.autosar.org/download/R4.0/AUTOSAR_SWS_CANDriver.pdf.

[26] Alexander Szakaly. Response time analysis with offsets for CAN. Master's thesis, Department of Computer Engineering, Chalmers University of Technology, Nov. 2003.

[27] Yang Chen, Ryo Kurachi, Hiroaki Takada, and Gang Zeng. Schedulability comparison for CAN message with offset: Priority queue versus FIFO queue. In *19th International Conference on Real-Time and Network Systems (RTNS), Sep., 2011*, pages 181–192, Sep. 2011.

[28] Lei Du and Guoqing Xu. Worst case response time analysis for CAN messages with offsets. In *IEEE International Conference on Vehicular Electronics and Safety (ICVES), Nov., 2009*, pages 41 –45, Nov. 2009.

[29] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst. System level performance analysis - the SymTA/S approach. *Computers and Digital Techniques,*, 152(2):148–166, March 2005.

[30] Patrick Yomsi, Dominique Bertrand, Nicolas Navet, and Robert Davis. Controller Area Network (CAN): Response time analysis with offsets. In *9th IEEE International Workshop on Factory Communication Systems (WFCS), May, 2012*, May 2012.

[31] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Extending schedulability analysis of Controller Area Network (CAN) for mixed (periodic/sporadic) messages. In *16th IEEE Conference on Emerging Technologies and Factory Automation (ETFA), Sep., 2011*, Sep. 2011.

[32] S. Mubeen, J. Mäki-Turja and M. Sjödin. Response-time analysis of mixed messages in Controller Area Network with priority- and FIFO-queued nodes. In *9th IEEE International Workshop on Factory Communication Systems (WFCS), May, 2012*, May 2012.

[33] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Response time analysis for mixed messages in CAN supporting transmission abort requests. In *7th IEEE International Symposium on Industrial Embedded Systems (SIES), June, 2012*, June 2012.

[34] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Extending response-time analysis of mixed messages in CAN with controllers implementing non-abortable transmit buffers. In *17th IEEE Conference on Emerging Technologies and Factory Automation (ETFA), Sep., 2012*, Sep. 2012.

[35] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Worst-case response-time analysis for mixed messages with offsets in Controller Area Network. In *17th IEEE Conference on Emerging Technologies and Factory Automation (ETFA), Sep., 2012*, Sep. 2012.

[36] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Extending offset-based response-time analysis for mixed messages in Controller Area Network. In *18th IEEE Conference on Emerging Technologies and Factory Automation (ETFA), Sep., 2013*, Sep. 2013.

[37] CANalyzer. 2013. http://www.vector.com/vi_canalyzer_en.html.

[38] CANoe, accessed in Oct., 2013. http://www.vector.com/portal/medien/cmc/info/CANoe_ProductInfor-mation_EN.pdf.

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

[40] RTaW-Sim. 2013. http://www.realtimeatwork.com/software/rtaw-sim.

[41] 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.

[42] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Communications-Oriented Development of Component- Based Vehicular Distributed Real-Time Embedded Systems. *Journal of Systems Architecture*, 2013.

[43] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Many-in-one Response-Time Analyzer for Controller Area Network. In *4th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, July 2013.

# Chapter 9

# Paper F: Support for end-to-end response-time and delay analysis in the industrial tool suite: Issues, experiences and a case study

Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin
In Journal of Computer Science and Information Systems (ComSIS), vol 10, nr 1, 2013.

**Abstract**

In this paper we discuss the implementation of the state-of-the-art end-to-end response-time and delay analysis as two individual plug-ins for the existing industrial tool suite Rubus-ICE. The tool suite is used for the development of software for vehicular embedded systems by several international companies. We describe and solve the problems encountered and highlight the experiences gained during the process of implementation, integration and evaluation of the analysis plug-ins. Finally, we provide a proof of concept by modeling the automotive-application case study with the existing industrial model (the Rubus Component Model), and analyzing it with the implemented analysis plug-ins.

**Keywords:** real-time systems, response-time analysis, end-to-end timing analysis, component-based development, distributed embedded systems.

## 9.1 Introduction

Often, an embedded system needs to interact and communicate 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. Many 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. The safety-critical nature of these systems requires evidence that the actions by them 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 these 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. Response Time Analysis (RTA) [1, 2] is one of the methods to check the schedulability of a system. It calculates upper bounds on the 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 task chains that may be distributed over several nodes in a Distributed Real-time Embedded (DRE) system.

A task chain is a sequence of more than one task in which every task (other than the first) receives a trigger, data or both from its predecessor. One way to classify these chains is as trigger chains and data chains. In trigger chains, there is only one triggering source (e.g, event, clock or interrupt) that activates the first task in the chain. The rest of the tasks are activated by their predecessors. In data chains, tasks are activated independent of each other, often with distinct periods. Each task (except the first) in these chains receives data from its predecessor. The first task in a data chain may receive data from the peripheral devices and interfaces, e.g., signals from the sensors or messages from the network interfaces. The end-to-end timing requirements on trigger chains are different from those on data chains. If a system is modeled with trigger chains only, it is called a single-rate system. On the other hand, if the system contains at least one data chain with different clocks then the system is said to be multi-rate.

In order to predict complete timing behavior of multi-rate real-time systems [6], the end-to-end delays should also be computed along with the holistic response times. For this purpose, the research community has developed the End-to-End Delay Analysis (E2EDA). In [6], the authors have a view that al-

most all automotive embedded systems are multi-rate systems. The industrial tools used for the development of such systems should be equipped with the state-of-the-art timing analysis.

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 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, HRTA and E2EDA in such a tool chain is non-trivial because there are several challenges that are encountered apart from merely coding and testing the analysis algorithms. These challenges and corresponding solutions that we propose are central to this paper.

### 9.1.1 Goals and Paper Contributions

In this paper, we discuss the implementation of holistic response time analysis and end-to-end [1] delay analysis as two plug-ins in the existing industrial tool suite Rubus-ICE (Integrated Component development Environment) [7]. 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 and end-to-end delay analysis to the existing tools for the industrial use.

2. Discuss and solve several problems encountered during the implementation, integration and evaluation of HRTA and E2EDA as two individual plug-ins for Rubus-ICE.

3. Discuss the experiences gained during the implementation, integration and evaluation of the HRTA and E2EDA plug-ins.

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

[1] The terms “holistic” and “end-to-end” mean the same thing. In order to be consistent with the previous work and naming conventions used in the existing industrial tools, we will use “holistic” with response-times and “end-to-end” with delays.

### 9.1.2 Paper Layout

The rest of the paper is organized as follows. Section 9.2 presents the background and related work. Section 9.3 discusses the end-to-end timing requirements and the analysis that we implemented in Rubus-ICE. Section 9.4 describes the challenges encountered, solutions proposed and experiences gained during the implementation and integration of the HRTA and E2EDA plug-ins. Section 9.5 presents our test plan. In Section 9.6, we present a case study by modeling and analyzing the automotive DRE application. Section 9.7 concludes the paper and presents the future work.

## 9.2 Background and Related Work

### 9.2.1 Relation to Authors' Previous Work

This work is the extension of our previous work [8] in which we discussed the implementation of only HRTA plug-in for the Rubus-ICE. In this paper, we implement E2EDA as a second plug-in. As compared to our previous work, this paper presents a detailed discussion on the end-to-end timing requirements in the industrial DRE systems. We also discuss the algorithm of end-to-end delay analysis and its conceptual organization in Rubus-ICE. Further, we discuss several challenging problems that were encountered during the implementation, integration and evaluation of the E2EDA plug-in. Moreover, we discuss the proposed solutions and gained experiences during the process of transferring state-of-the-art research results to the industrial tool suite. For the sake of completeness, we also revisit the problems and their solutions corresponding to the HRTA plug-in.

We also reconducted the case study. This is because the automotive DRE application (Autonomous Cruise Control system) considered in the previous work was modeled with only trigger chains. This limited the usability of our modeling and analysis tools because many automotive embedded systems in the industry are build using data and mixed chains as well. Therefore, we remodeled the same automotive-application with trigger, data and mixed chains. We also analyzed it with both the HRTA and E2EDA plug-ins. With the addition of E2EDA plug-in, a complete end-to-end timing analysis of DRE systems can be performed. Thus, the scope and usability of Rubus tools has widened with the addition of HRTA and E2EDA plug-ins.

### 9.2.2 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 [7] in close collaboration with several academic and industrial partners. Rubus is today mainly used for development of control functionality in vehicles by several international companies [9, 10, 11, 12]. The Rubus concept is based around the Rubus Component Model (RCM) [13] 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 the SWC 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.

RCM separates the control flow from the data flow among SWCs within a node. Thus, explicit synchronization and data access are visible at the modeling level. One important principle in RCM is to separate functional code and infrastructure implementing the execution model. RCM facilitates analysis and reuse of components in different contexts (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. Recently, we extended RCM for the development of DRE systems by introducing new components [14, 15, 16]. A detailed comparison of RCM with several component models is presented in [15].

Figure 9.1 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 the Intermediate Compiled Component Model (ICCM). After that the builder tool sequentially runs a set of plug-ins. Finally, a coder tool generates the code.

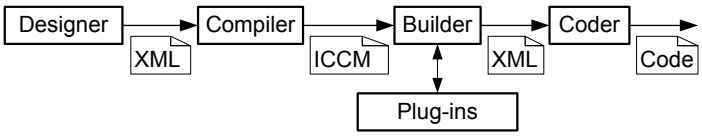


Figure 9.1: Sequence of steps from design to code generation in Rubus-ICE

### 9.2.3 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 response time analysis and shared stack analysis.

### 9.2.4 Plug-in Framework in Rubus-ICE

The plug-in framework in Rubus-ICE [17] facilitates the implementation of state-of-the-art research results in 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 the builder tool as shown in Figure 9.1. 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 inputs, runs to completion and finally writes the results to the ICCM file. The 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 9.2 shows a conceptual organization of a Rubus-ICE plug-in.

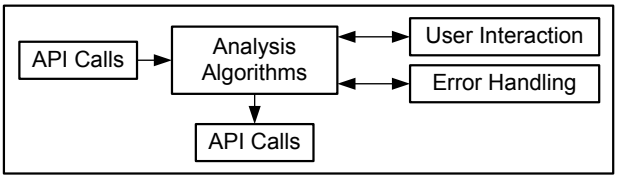


Figure 9.2: Conceptual organization of a plug-in in Rubus-ICE

### 9.2.5 Response-Time Analysis

**RTA of Tasks in a Node.**

Liu and Layland [18] provided theoretical foundation for analysis of fixed-priority scheduled systems. Joseph and Pandya published the first RTA [19] for the simple task model presented in [18]. 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 [20]. In [20], 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.

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 [21] reduced pessimism from RTA developed in [4, 5] and presented a tighter RTA for tasks with offsets by accurately modeling inter-task interference. In [6, 22], the authors point out that the existing RTA does not target general multi-rate systems. We implemented tighter version of RTA of tasks with offsets [21] as part of the end-to-end response-time and delay analysis.

**RTA of Messages in a Network.**

There are many protocols such as Controller Area Network (CAN), Time-Triggered CAN (TTCAN), FlexRay, etc., that are used for network communication in DRE systems. To stay focussed on the automotive or vehicular domain, we will consider only CAN and its high-level protocols. Tindell et al. [23] 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. [24].

The analysis in [23, 24] assumes that all CAN device drivers implement priority-based queues. In [25] 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. In this work, the message deadlines are assumed to be smaller than or equal to the corresponding periods. In [26], Davis et al. lifted this assumption by supporting the analysis of CAN messages with arbitrary deadlines. Furthermore, they extended their work to support RTA of CAN for FIFO and work-conserving queues.

However, the existing analysis does not support mixed messages which are implemented by several high-level protocols for CAN. In [27, 28, 29], 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. Later on, Mubeen et al. [30] extended the existing analysis for CAN to support mixed messages that are scheduled with offsets in the controllers that implement priority-ordered queues. In this work we will consider all of the above analyses as part of the end-to-end response-time and delay analysis.

**Holistic RTA.**

It combines the analysis of nodes (uniprocessors) and networks. In this paper, we consider the end-to-end timing model that corresponds to the holistic schedulability analysis for DRE systems [3]. In [31], Pop et al. provide a holistic schedulability analysis of distributed embedded systems in which tasks are both time- and event-triggered. The analysis is developed for ST/DYN bus protocol that uses static and dynamic phases for sending messages. As compared to this approach, we implement the holistic analysis of [3] because it provides the flexibility to use several network-communication protocols used in the automotive domain. In [32], we discussed our preliminary findings about implementation issues that are encountered when HRTA is transferred to the industrial tools.

**End-to-end Delay Analysis.**

Stappert et al. [22] formally described end-to-end timing constrains for multi-rate systems in the automotive domain. In [6], Feiertag et al. presented a framework (developed in TIMMO project [33]) for the computation of end-to-end delays for multi-rate automotive embedded systems. Furthermore, they emphasized on the importance of two end-to-end latency semantics, i.e., "maximum age of data" and "first reaction" in control systems and body electronics domains respectively. A scalable technique, based on model checking, for the computation of end-to-end latencies is described in [34]. In this work, we will implement the end-to-end delay analysis [6] as a plug-in for the Rubus-ICE tool suite.

### 9.2.6 Tools for End-to-end Timing Analysis of DRE Systems

We briefly discuss few tool suites that provide similar real-time analysis support for DRE systems. The MAST tool suite [35] 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 [21] is implemented as part of the end-to-end response-time and delays analysis in Rubus-ICE. The MAST model also allows visual modeling and analysis of real-time systems in a Unified Modeling Language (UML) design environment.

The Volcano Family [36] is a bunch of tools for designing, analyzing, testing and validating automotive embedded software systems. Among them, Volcano Network Architect (VNA) [37] is a communication design tool that supports the analysis of Local Interconnect Network (LIN) and CAN networks. It also supports end-to-end timing analysis of a system with more than one network. It implements RTA of CAN developed by Tindell et al. [23].

SymTA/S [38] 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 bus 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 [39].

Vector [40] is a tools provider for the development of networked electronic systems in the automotive and related domains. In the Vector tool family, CANoe [41] is a tool for the development, testing and analysis of ECU (Electronic Control Units) networks and individual ECUs. It supports various protocols for network communication including CAN, LIN, MOST, Flexray, Ethernet

and J1708. Network Designer CAN is another tool by Vector that is used to design the architecture and perform timing analysis of CAN network.

RAPID RMA [42] 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 [43].

The Rubus-ICE tool suite allows a developer to specify timing information and perform end-to-end response time and delay analysis at the modeling phase during component-based development of DRE systems. To the best of our knowledge, Rubus-ICE is the first and only tool suite that implements RTA of mixed messages in CAN [27], RTA of mixed messages with offsets [30] and a tighter version of offset-based RTA algorithm [21] as part of the end-to-end response time and delay analysis.

## 9.3 End-to-end Timing Requirements and Implemented Analysis in Rubus-ICE

### 9.3.1 End-to-end timing requirements in trigger chains

A real-time system can be modeled with trigger chains (see Figure 9.4 and Figure 9.5), data chains (see Figure 9.6 and Figure 9.8) or a combination of both. The end-to-end timing requirements on trigger chains are different from those on data chains. If the system is modeled with trigger chains then the interest, from the schedulability point of view, lies in the calculation of end-to-end or holistic response times and their comparison with corresponding deadlines. Hence, end-to-end deadline requirements placed on trigger chains correspond to their holistic response times. If holistic response times of all trigger chains are less than or equal to corresponding deadlines, the system is considered schedulable.

The holistic response-time analysis calculates the response times of event chains that are distributed over several nodes (also called distributed transactions) in a DRE system. An example of a distributed transaction in a DRE system is shown in Figure 9.3. The holistic response time is equal to the elapsed time between the arrival of an event (corresponding to the brake pedal input in the sensor node) and the response time of Task4 (corresponding to the production of brake actuation signal in the actuation node).

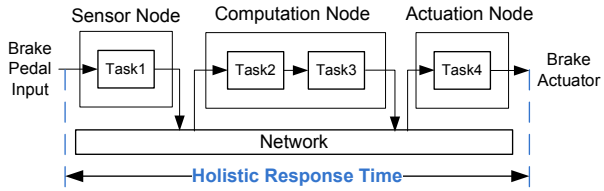


Figure 9.3: Holistic response-time in a distributed real-time system

**Examples.**

An example of a trigger chain that consists of three components is shown in Figure 9.4. Assume that each component corresponds to a task at run-time. When task $\tau_{SWC\_A}$ finishes its execution, it triggers $\tau_{SWC\_B}$. Similarly, $\tau_{SWC\_C}$ can only be triggered by $\tau_{SWC\_B}$ after finishing its execution. There cannot be multiple outputs corresponding to a single input signal. In fact, there will always be one output of the chain corresponding to the input trigger. Hence, the end-to-end timing requirements correspond to the holistic response times. In order to provide a comparison of holistic response time in a trigger chain with the end-to-end delays in a data chain, assume that the trigger chain shown in Figure 9.4 is the only chain of tasks in the system. Let the priorities of all tasks be the same while WCET of each task is $1ms$. The holistic response time of this trigger chain is equal to the response time of $\tau_{SWC\_C}$ which is, intuitively, equal to $3ms$.

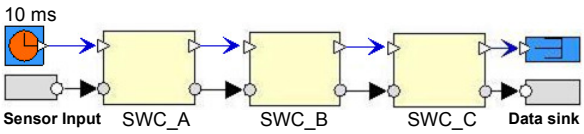


Figure 9.4: RCM model of a trigger chain in a single-node real-time system

Distributed real-time systems can also be modeled with trigger chains. Consider a model of a two-node distributed real-time system modeled with RCM as shown in Figure 9.5. There is only one triggering ancestor in node A that activates $SWC\_A$ which, in turn, triggers $OSWC\_A$ component that is responsible for sending a message to CAN. The $ISWC\_C$ component in only activated when an interrupt is raised due to the arrival of a CAN message at node C. Hence, these three components form a distributed trigger chain. Once

again, the end-to-end timing requirements correspond to the holistic response times.

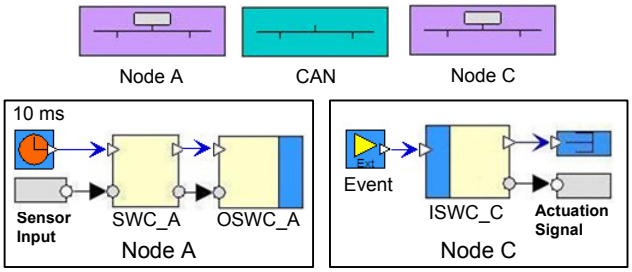


Figure 9.5: RCM model of a distributed trigger chain in a DRE system

### 9.3.2 End-to-end timing requirements in data chains

As compared to the systems which are modeled with trigger chains, merely computing the holistic response times and comparing them with the end-to-end deadlines is not sufficient to predict the complete timing behavior of multi-rate real-time systems which are modeled with data chains. There may be over- and under-sampling in such systems because the individual tasks are activated by independent clocks, often with different periods. Since data is transferred among tasks and messages within a data chain by means of asynchronous buffers, there exist different semantics of end-to-end delay in a data chain. These buffers are often of a non-consuming type which means the data stays in the buffer after it is read by the reader task. Moreover, the data in the buffer can be overwritten by the writer task with new values before the previous value was read by the reader task. Therefore, some input values in the data buffers can be overwritten by new values, and hence the effect of the old input values may never propagate to the output of a data chain. Further, it is also possible to have several duplicates of the output of a data chain corresponding to a particular input.

The end-to-end timing requirements in multi-rate real-time systems, especially in the automotive domain, are placed on the first reaction to the input and age of the data received at the output [6]. Hence, it is important to calculate the end-to-end delays in these systems. The end-to-end delay in a data chain refers to the time elapsed between the arrival of a signal at the first task and production of corresponding output signal by the last task in the chain (provided the

information corresponding to the input signal has traversed the chain from first to last task) [34].

In a single-rate real-time system that contains only trigger chains, tasks in a chain are not activated by independent events, in fact, there is only one activating event in the chain. Hence, the holistic response times and end-to-end delays will have equal values. On the other hand, these values are not the same in multi-rate real-time systems that are modeled with data chains. Therefore, a complete analysis of a real-time system modeled with data chains requires the calculation of not only holistic response times but also end-to-end delays.

**Examples.**

A multi-rate real-time system modeled with three SWCs in RCM is shown in Figure 9.6. These SWCs are activated by independent clocks with different periods, i.e., 8ms, 16ms and 4ms respectively. $SWC\_A$ reads the input signals from the sensors while $SWC\_C$ produces the output signals for the actuators. Assume that each SWC will be allocated to an individual task by the run-time environment generator. Also assume that WCET of each task is $1ms$.

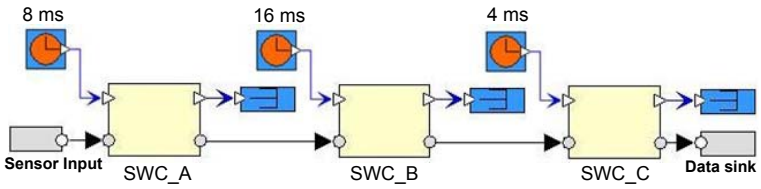


Figure 9.6: RCM model of a data chain in a single-node real-time system

The time line corresponding to the run-time execution of the three tasks (corresponding to three SWCs) is depicted in Figure 9.7. It can be seen that there are multiple outputs corresponding to a single input signal. The four end-to-end delays are identified in Figure 9.7.

**Last In First Out (LIFO).** This delay is equal to the time elapsed between the current non-overwritten release of task $\tau_A$ (input of the data chain) and corresponding first response of task $\tau_C$ (output of the data chain).

**Last In Last Out (LILO).** This delay is equal to the time elapsed between the current non-overwritten release of task $\tau_A$ (input of the data chain) and

corresponding last response of task $\tau_C$ (output of the data chain). This delay is identified as "Data Age"[2] in [6]. Data age specifies the longest time data is allowed to age from production by the initiator until the data is delivered to the terminator. This delay finds its importance in control applications where the interest lies in the freshness of the produced data. For a data chain in a control system that initiates with a sensor input and terminates by producing an actuation signal, it is very important to ensure that the actuator signal does not exceed a maximum age [6].

Generally speaking, we consider the last non-overwritten input that actually propagates through the data chain towards the output in the case of both LIFO and LILO delays.

**First In First Out (FIFO).** This delay is equal to the time elapsed between the previous non-overwritten release of task $\tau_A$ (input of the data chain) and first response of task $\tau_C$ (output of the data chain) corresponding to the current non-overwritten release of task $\tau_A$. Assume that a new value of the input is available in the input buffer of task $\tau_A$ "just after" the release of the second instance of task $\tau_A$ (at time $8ms$). Hence, the second instance of task $\tau_A$ "just misses" the read of the new value from its input buffer. This new value has to wait for the next instance of task $\tau_A$ to travel towards the output of the data chain. Therefore, the new value will be read by the third and forth instances of task $\tau_A$. The first output corresponding to the new value (arriving just after $8ms$) will appear at the output of the chain at $34ms$. This will result in the FIFO delay of $26ms$ as shown in Figure 9.7. This phenomenon is more obvious in the case of distributed embedded systems where a task in the receiving node may just miss to read fresh signals from a message that is received from the network.

This delay is identified as "first reaction to data or Data Reaction"[3] in [6]. Data reaction delay is the longest allowed reaction time for data produced by the initiator to be delivered to the terminator. This delay finds its importance in the button-to-reaction applications in body electronics domain where first reaction to input is important.

**First In Last Out (FILO).** This delay is equal to the time elapsed between the previous non-overwritten release of task $\tau_A$ (input of the data chain) and last response of task $\tau_C$ (output of the data chain) corresponding to the current

[2] We will use the term "Data Age delay" to refer to LILO delay throughout the paper.

[3] We will use the term "Data Reaction delay" to refer to FIFO delay throughout the paper.

non-overwritten release of task $\tau_A$. The reasoning about "just missing" a fresh input that we discussed in the case of FIFO delay is also applicable in the case of FILO delay.

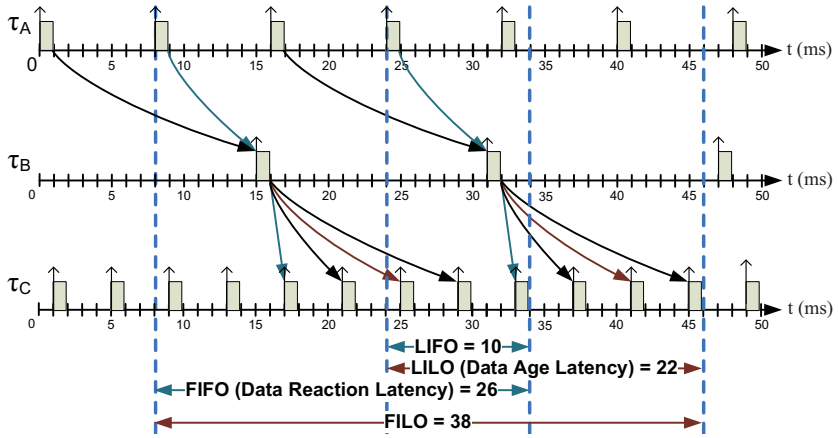


Figure 9.7: End-to-end delays for a data chain in a real-time system

In the case of distributed real-time systems, data chains may also be distributed over more than one node. Consider a model of a two-node distributed real-time system modeled with RCM as shown in Figure 9.8. The nodes are connected to the CAN network. The internal model of the nodes is also shown in Figure 9.8. In Node A, $SWC\_A$ is triggered by a clock with a period of 8ms. The $OSWC\_A$ component that is responsible for sending a message to the network is triggered by another clock with a period of 16ms. $ISWC\_C$ is a component that receives a message from the network and is activated by a clock with a period of 4ms. Assume that each component is allocated to a separate task at run-time, i.e., the components $SWC\_A$, $OSWC\_A$ and $ISWC\_C$ are allocated to tasks $\tau_A$, $\tau_B$ and $\tau_C$ respectively. Since, the system consists of tasks with similar activation patterns and periods as compared to the tasks in the single-node real-time system example discussed above, it can be scheduled in a similar manner as indicated by $\tau_A$, $\tau_B$ and $\tau_C$ in Figure 9.7. The end-to-end delays are also defined in a similar fashion.

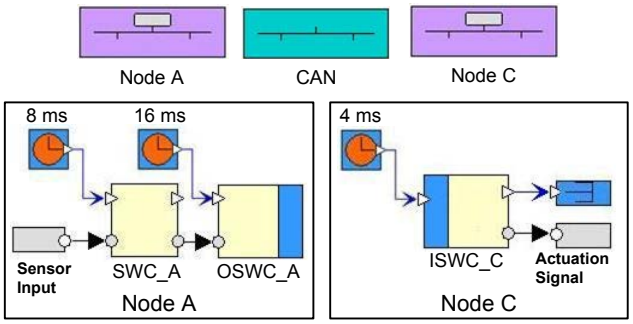


Figure 9.8: RCM model of a distributed data chain in a DRE system

### 9.3.3 Implemented Holistic Response-Time Analysis

We implemented HRTA as a 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 real-time system.

In order to analyze tasks in each node, we implement RTA of tasks with offsets developed by [4, 5] and improved by [21]. We implement the 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 [23, 24].
2. RTA of CAN for mixed messages [27].
3. RTA of CAN for mixed messages with offsets [30][4].

The above analyses assume that 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 and work-conserving queues [26, 25] and RTA of CAN with FIFO Queues for Mixed Messages [28].

The pseudocode of HRTA algorithm is shown in Algorithm 3. The HRTA algorithm iteratively runs the algorithms for node and network analyses. 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

[4]The analysis of this profile is implemented as a standalone analyzer whose integration with Rubus-ICE is a work in progress

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 difference between the worst- and best-case response times of its sender task (computed in the first step). Similarly, each task that receives the message inherits a release jitter equal to the difference between the worst- and best-case response times 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 we implemented in Rubus-ICE is shown in Figure 9.9.

**Algorithm 3** Algorithm for holistic response-time analysis

```
1: begin
2: RT_Prev ← 0                          ▷ Initialize all Response Times (RTs) to zero
3: Repeat ← TRUE
4: while Repeat = TRUE do
5:     for all Messages_and_tasks_in_the_system do
6:         Jitter_Msg ← (WCRT_Sender_task − BCRT_Sender_task)   ▷ WCRT:
   Worst-Case Response Time, BCRT: Best-Case Response Time
7:         Jitter_Receiver_task ← (WCRT_Msg − BCRT_Msg)
8:         COMPUTE_RT_OF_ALL_MESSAGES()
9:         COMPUTE_RT_OF_ALL_TASKS_IN_EVERY_NODE()
10:        if RT > RT_Prev then
11:            RT_Prev ← RT
12:            Repeat ← TRUE
13:        else
14:            Repeat ← FALSE
15:        end if
16:    end for
17: end while
18: end
```

### 9.3.4 Implemented End-to-end Delay Analysis

We implemented the end-to-end delay analysis that is derived in [6] as the E2EDA plug-in for Rubus-ICE. This analysis implicitly requires the calculation of response times of individual tasks, messages and holistic response times

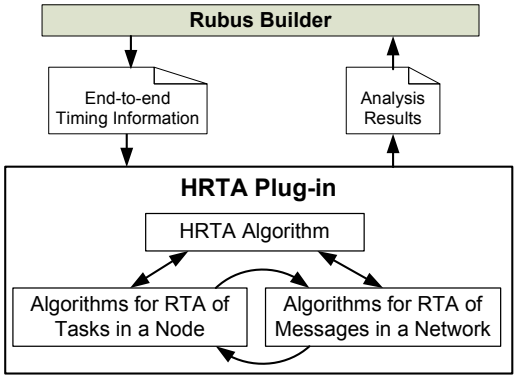


Figure 9.9: Conceptual view of the HRTA plug-in in Rubus-ICE

of task chains. For example, the calculation of four end-to-end delays for the multi-rate real-time system shown in Figure 9.6 requires the response time of the task $\tau_C$ (corresponding to the component $SWC\_C$) and the activation times of tasks $\tau_A$ and $\tau_C$. Similarly, the calculation of four end-to-end delays for the multi-rate DRE system shown in Figure 9.8 requires the calculation of the response time of the task $\tau_C$ in node C and the activation times of tasks $\tau_A$ and $\tau_C$ in nodes A and C respectively. Since, the HRTA plug-in is able to calculate response times of tasks, network messages and task chains, we reuse the analysis results computed by the HRTA plug-in as an input to the E2EDA plug-in as shown in Figure 9.10. The pseudocode of E2EDA algorithm[5] is shown in Algorithm 4.

## 9.4 Encountered Problems, Proposed Solutions and Gained Experiences

In this section we discuss several problems encountered during the process of implementation and integration of HRTA and E2EDA as plug-ins for the Rubus-ICE tool suite. We also present our solution to each individual problem. Moreover, we discuss the summary of the experiences that we gained while transferring theoretical research results (i.e., HRTA and E2EDA) to the industrial tool suite.

[5] [6] should be referred for detailed analysis.

**Algorithm 4** Algorithm for end-to-end delay analysis

**begin**
GET_RT_OF_ALL_TASKS_MESSAGES_TASK_CHAINS() ▷ Get the analysis results from the HRTA plug-in
FIND_ALL_VALID_TIMED_PATHS() ▷ Timed Path (TP) is a sequence of task instances from input to output. A TP is valid if information flow among tasks is possible [6], e.g., [$\tau_A(1^{st}$instance), $\tau_B(1^{st}$instance), $\tau_C(5^{th}$instance)] in Figure 9.7 is a valid TP. On the other hand, TP [$\tau_A(1^{st}$instance), $\tau_B(1^{st}$instance), $\tau_C(1^{st}$instance)] in Figure 9.7 is invalid because information cannot flow between $\tau_B(1^{st}$instance) and $\tau_C(1^{st}$instance)
**procedure** COMPUTE_FF_DELAY(FF_TP)
    FF_delay = $\alpha_n$(instance) + $\delta_n$(instance) - $\alpha_1$(instance) ▷ $\alpha_n$(instance): Activation time of the corresponding instance of the n$^{th}$ task in timed path FF_TP
    ▷ $\delta_n$(instance): Response time of the corresponding instance of the n$^{th}$ task in timed path FF_TP
    **return** FF_delay
**end procedure**
    ▷ The above mentioned procedure calculates $FF_{Delay}$ only. [6] should be referred for the calculation of the rest of the delays
**for all** Delay_constraints_specified_in_the_system **do**
    $FF_{Delay} \leftarrow 0, FL_{Delay} \leftarrow 0, LF_{Delay} \leftarrow 0, LL_{Delay} \leftarrow 0$ ▷ Initialize all delays
    COMPUTE_ALL_REACHABLE_TIMED_PATHS() ▷ All those paths from input to output in which the changes in input actually travel towards the output, e.g., [$\tau_A(2^{nd}$instance), $\tau_B(1^{st}$instance), $\tau_C(5^{th}$instance)] in Figure 9.7
    $FF\_TP_{count} \leftarrow$ GET_ALL_FF_TPS() ▷ TP: Timed Path, FF: First to First
    $FL\_TP_{count} \leftarrow$ GET_ALL_FL_TPS() ▷ FL: First to Last
    $LF\_TP_{count} \leftarrow$ GET_ALL_LF_TPS() ▷ LF: Last to First
    $LL\_TP_{count} \leftarrow$ GET_ALL_LL_TPS() ▷ LL: Last to Last
    **for** i:=1 **do**$FF\_TP_{count}$
        **if** COMPUTE_FF_DELAY(i) $> FF_{Delay}$ **then**
            $FF_{Delay} \leftarrow$ COMPUTE_FF_DELAY()
        **end if**
    **end for**

```
20:     for i:=1 do FL_TP_count
21:         if COMPUTE_FL_DELAY(i) > FL_Delay then
22:             FL_Delay ← COMPUTE_FL_DELAY()
23:         end if
24:     end for
25:     for i:=1 do LF_TP_count
26:         if COMPUTE_LF_DELAY(i) > LF_Delay then
27:             LF_Delay ← COMPUTE_LF_DELAY()
28:         end if
29:     end for
30:     for i:=1 do LL_TP_count
31:         if COMPUTE_LL_DELAY(i) > LL_Delay then
32:             LL_Delay ← COMPUTE_LL_DELAY()
33:         end if
34:     end for
35: end for
36: end
```

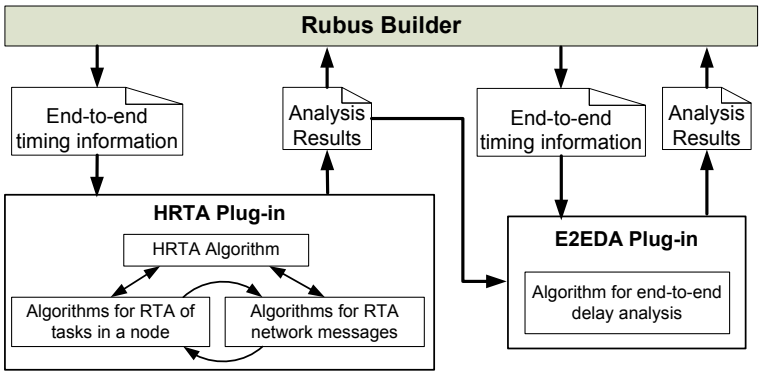


Figure 9.10: Conceptual view of the E2EDA plug-in in Rubus-ICE

### 9.4.1 Extraction of Unambiguous Timing Information

One common assumption in end-to-end response time and delay analyses is that the timing attributes required by the analysis are available as input. However, when these analyses are implemented in a tool chain used for the component-based development of 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 often build upon different meta-models [44]. Moreover, the design model can contain redundant timing information. Hence, it is not trivial to extract unambiguous timing information for HRTA and E2EDA.

We divide the timing information (to be extracted) into two categories.

**Extraction of Timing Information Corresponding to User Inputs.**

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 task chains, jitters, etc. In [16], 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 the end-to-end response time and delay analysis.

**Extraction of Timing Information from the Modeled Application.**

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 the case of periodic transmission) or message inhibit time (in the 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 it. Thus, we assign period or inhibit time to the message which is 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 this 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 problem can be visualized in the example shown in Figure 9.11. It should be noted that the 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 [15].

In Figure 9.11(a), the sender task is activated by a clock, and hence the corresponding message is periodic. Similarly, the corresponding message is sporadic in Figure 9.11(b) because the sender task is activated by an event. However, the sender task in Figure 9.11(c) is triggered by both a clock and an event. Here the relationship between two triggering sources is important. If there exists a dependency relation between them as in the case of mixed transmission mode in the CANopen protocol [45] and AUTOSAR communication [46] then such message will be considered as a special type of sporadic message. On the other hand, if triggering sources are independent of each other as in the case of implementation in the HCAN protocol [47] then the corresponding message will be considered a mixed message [27, 28].

If there are periodic and sporadic messages in the modeled application, the HRTA plug-in uses the first profile for network analysis (discussed in Section 3.3). On the other hand, if the modeled application contains mixed messages as well, the second profile for 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 periodic or sporadic transmission type respectively. However, if the sender is activated periodically as well as sporadically and both triggering sources are independent of each other, the message is assigned the mixed transmission type.

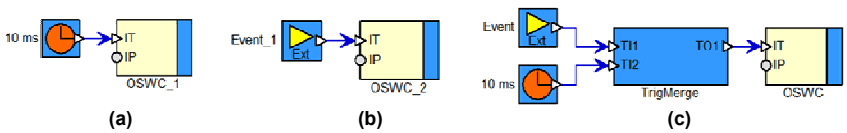


Figure 9.11: Extraction of transmission type of a message

**Identification of Trigger, Data and Mixed Chains.** The end-to-end timing requirements on trigger chains are different from those on data chains. These requirements correspond to end-to-end response times for trigger chains and both end-to-end response times and delays for data chains. Data and trigger chains should be distinctly identified and the corresponding timing requirements should be unambiguously captured in the timing model on which the analysis tools operate. For this purpose, we add a new attribute “trigger dependency” in the data structure of tasks in the analysis model. If a task is

triggered by an independent source such as a clock then this attribute will be assigned "independent". On the other hand, if the task is triggered by another task then this parameter will be assigned "dependent". Moreover, a precedence constraint will also be specified on this task in the case of dependent triggering. This is because, a task in a trigger chain cannot start its execution before the completion of the previous task in the chain.

However, a system can also be modeled with task chains that are comprised of data chains as well as trigger chains. We call these chains as mixed chains. An example of a mixed chain modeled with RCM is shown in Figure 9.12. In this chain, components $SWC\_A$, $SWC\_B$ and $SWC\_E$ are triggered by independent clocks and which is the property of components in a data chain. Hence, the "trigger dependency" attribute of the tasks corresponding to these three components will be assigned "independent". Whereas, the components $SWC\_C$ and $SWC\_D$ are triggered by their respective predecessors and which is the property of components in a trigger chain. The "trigger dependency" attribute of the tasks corresponding to these two components will be assigned "dependent".

A task chain is identified by checking the "trigger dependency" parameter for each individual task in the chain. If this parameter is "dependent" for all tasks (except the first or initiating task) then the chain is identified as a trigger chain. On the other hand, if this parameter for each task in the chain is "independent" then the chain is identified as a data chain. However, if this parameter for some tasks is "independent" and for the others it is "dependent" then the chain is considered as mixed.

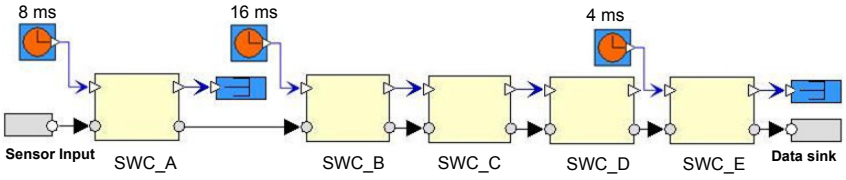


Figure 9.12: RCM model of a mixed chain in a single-node real-time system

The problem of identification of a task chain becomes more challenging to resolve when a chain mimics as a data chain as well as a trigger chain by means of trigger merges as shown in Figure 9.13. It can be seen that $SWC\_C$ component can be triggered by both its predecessor task and a clock. In this case, the "trigger dependency" attribute is assumed to have both the values, i.e., "independent" and "dependent". If such task is identified in a task chain,

we consider it as a special type of mixed chain. For this chain, the end-to-end timing requirements correspond to both holistic response times and end-to-end delays.

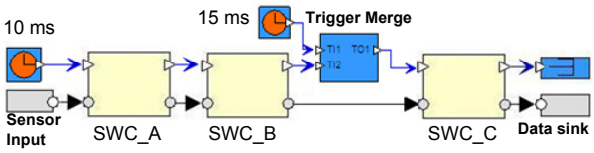


Figure 9.13: An example of a data chain with trigger merges

### 9.4.2 Extraction of Linking Information from Distributed Transactions

In order to perform HRTA, correct linking information of DTs should be extracted from the design model [48]. For this purpose, we need to have a mapping among signals, data ports and messages in the system. Consider the following DT in a two-node DRE system modeled with RCM as shown in Figure 9.14.

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

In this example, our focus is on the network interface components, i.e., OSWC and ISWC [15]. In order to compute the holistic response time of this DT, we need to extract linking information from the component model. We identified the 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 the bidirectional arrow in Figure 9.14.

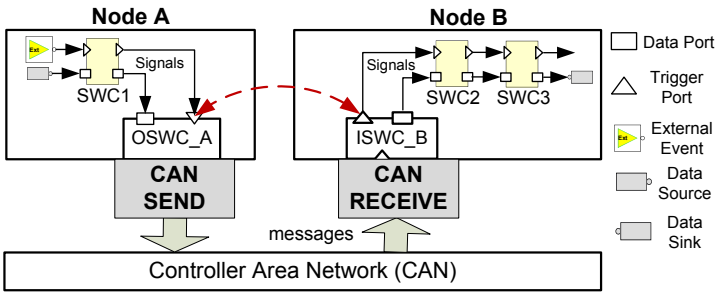


Figure 9.14: Two-node DRE system modeled with RCM

Since, the E2EDA plug-in needs to compute all valid timed paths (i.e., those paths in which input actually travels to the output) from initiator to the terminator for every data chain (see Algorithm 2), the linking information among all tasks and messages in the data chain should be available. We extract this information about all tasks and messages in those data chains on which end-to-end delay constraints are specified. The linking information also includes Trigger Dependency attribute for every task in the chain.

### 9.4.3 Analysis of Distributed Transactions with Branches

If a modeled DRE application contains branches of task chains that are distributed over several nodes and have one common terminator task, the calculations for the end-to-end response-time and delays of such chains are not straight forward. Consider the example of a two-node DRE system containing branches in DTs as shown in Figure 9.15. The components $OSWC\_A1$ and $OSWC\_A2$ in node A send messages $m1$ and $m2$ which are received by the components $ISWC\_C1$ and $ISWC\_C2$ in node C respectively. Hence, there are two DTs that have different initiators but a single terminator, i.e., $SWC\_C3$. These transactions are listed below.

1. $SWC\_A1 \rightarrow SWC\_A2 \rightarrow OSWC\_A1 \rightarrow ISWC\_C1 \rightarrow SWC\_C1 \rightarrow SWC\_C3$
2. $SWC\_A3 \rightarrow OSWC\_A2 \rightarrow ISWC\_C2 \rightarrow SWC\_C2 \rightarrow SWC\_C3$

Assume that Data Age delay constraint is specified on $SWC\_C3$. Also assume that the start of this constraint is specified on the component $SWC\_A1$ in node A. Therefore, we need to perform end-to-end delay analysis only on the first DT (in the above list). It should be noted that the start (initiating task of the data chain) and end (terminating task of the data chain) of each delay constraint should be specified by the user. We know from Section 3 (Algorithm 2) that the calculations for Data Age delay require the calculation of the holistic response time, i.e., the response time of the last task in the chain (task corresponding to $SWC\_C3$ component). However, the response time of this task depends upon the the holistic response times of both DTs listed above. In this case, the HRTA plug-in will calculate the holistic response times of all branches (two in this case) while the E2EDA plug-in will consider the maximum value among these holistic response times during calculations for the end-to-end delays. Although, the example in Figure 9.15 consisted of data chains only, the HRTA plug-in treats trigger chains in a similar fashion.

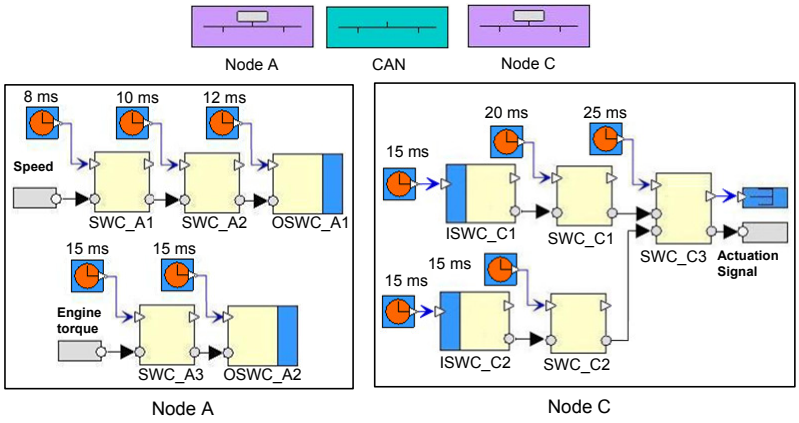


Figure 9.15: RCM model of a two-node DRE system with branches in distributed transactions

### 9.4.4 Analysis of Mixed Task Chains

The four different end-to-end delays (discussed in Section 3.2) do not exist in the case of trigger chains. This is because trigger chains are analogous to single-rate systems in which over- and under-sampling cannot occur. More-

over, there can never be multiple copies of a single input. If the user specifies end-to-end delay constraints on a trigger chain then the E2EDA plug-in detects this incorrect specification and complains about it. However, a system can also be modeled with mixed chains that are comprised of data chains as well as trigger chains as shown in Figure 9.12. Although a mixed chain contains a trigger chain, it is meaningful to compute both holistic response time as well as end-to-end delays for it. Therefore, the newly developed plug-ins compute the holistic response-times as well as end-to-end delays for mixed chains.

There are two options to handle mixed chains in the analysis model. In the first option, if a component is triggered by its predecessor then it is assumed to be triggered by independent clock with the same period as that of its predecessor's clock. Moreover, this component is assumed to have implicit precedence relation with its predecessor, i.e., it can be executed only upon completion of its predecessor's execution. $SWC\_C$ and $SWC\_D$ are the examples of such components in the mixed chain shown in Figure 9.12. Using this option, the execution time line of the task chain corresponding to component chain of Figure 9.12 is shown in Figure 9.16. This time line will be used by the E2EDA plug-in to compute the total number of timed paths. However, there are several timed paths (indicated with crosses in Figure 9.16) that are impossible to occur in reality. This is because each instance of a task in a trigger chain can be triggered only by one instance of its predecessor task. This will result in unnecessary calculations, i.e., a considerable overhead on the execution time of the analysis plug-ins. Therefore, we do not implement this option in the analysis model.

Instead, we use the second option that reduces the mixed chain by combining all tasks belonging to a trigger sub-chain (within the mixed chain) into a single task activated by independent clock. Hence, the reduced mixed chain resembles a data chain. For example, $SWC\_B$, $SWC\_C$ and $SWC\_D$ are combined to a single task (with combined WCETs, offsets, etc.) which is triggered by independent clock whose period is exactly the same as that of the clock that triggers $SWC\_B$ component. The execution time line of the task chain corresponding to reduced mixed chain of Figure 9.12 is shown in Figure 9.17. The corresponding end-to-end delays are also depicted in Figure 9.17. By implementing the second option , we got rid of the so-called "impossible timed paths". It should be noted that these chain reductions are not required by the HRTA plug-in. Mixed chain reduction method is only applied in the analysis model of the E2EDA plug-in.

Mixed chains may also exist in the models of DRE systems where they may contain many combinations of data and trigger chains distributed over several

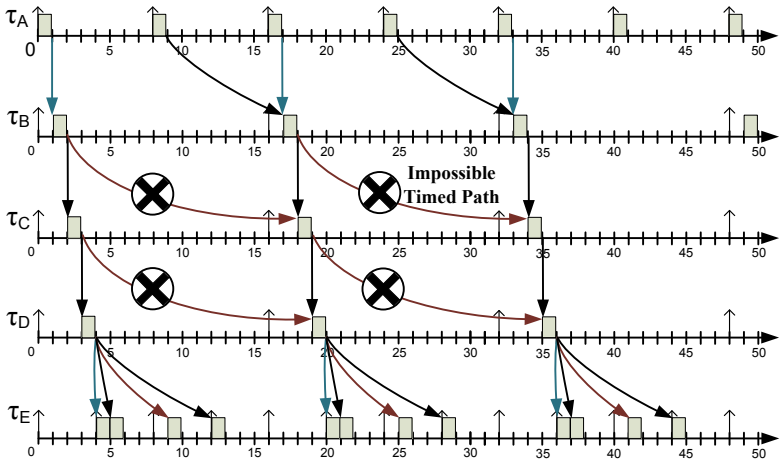


Figure 9.16: Demonstration of impossible timed paths in mixed chains

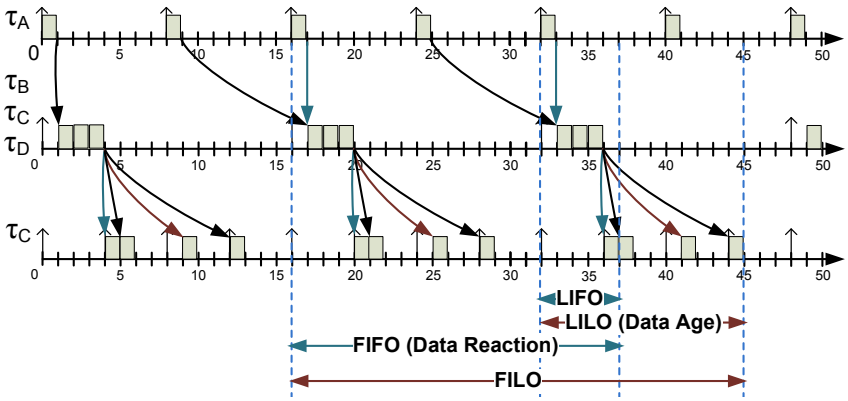


Figure 9.17: Reduction of a mixed chain in the analysis model

nodes. Four such combinations in a two-node DRE system are shown in Figure 9.18. Mixed chain reduction method is applied on distributed mixed chains in a similar fashion.

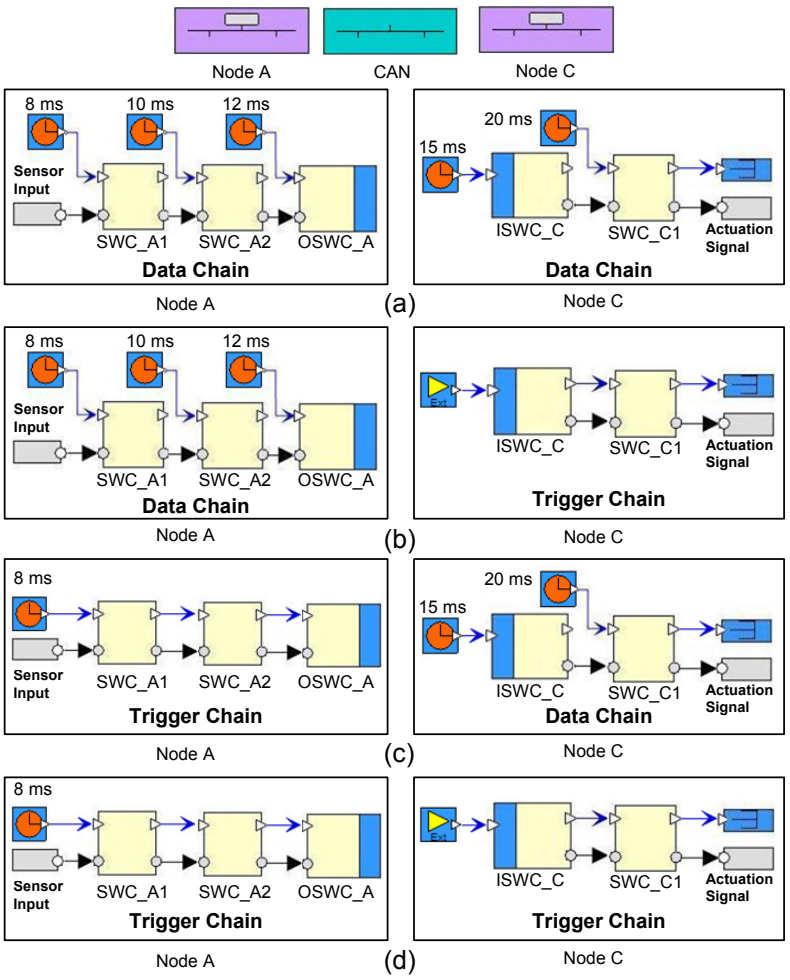


Figure 9.18: Different combinations of data and trigger chains in a two-node DRE system modeled with RCM

### 9.4.5 Analysis of the System Containing Messages Received from Outside of the Model

One of the requirements by the users of the analysis tools was that the HRTA and E2EDA plug-ins should be able to support the analysis of a system that

receives messages from unknown senders (from outside of the modeled application). One motivation behind this requirement may be the integration of two systems that are build using different methodologies and tools. Second motivation could be the integration of legacy systems with newly developed systems. Another motivation could be the requirement for the end-to-end timing analysis early during the development. At early stage, the models of some nodes may not be available. However, the signals and messages which these missing nodes are supposed to send and receive might have been decided. With these requirements on the system development, the network is assumed to contain messages whose sender nodes are not developed yet. Similarly, the available nodes may send messages via network to the nodes that will be available at a later stage.

As we discussed earlier in Section 3.3, the holistic response-time analysis connects the tasks and messages in a DT by means of attribute inheritance [3]. This means that a message inherits the difference between the worst- and best-case response times of the sending task as its release jitter. Moreover, the message also inherits other attributes from the sender task such as transmission type (periodic, sporadic or mixed [27]); and period or inhibit time or both. Since, the HRTA algorithm is iterative, the attribute inheritance is repeatedly carried out until holistic response time of the chain converges or corresponding deadlines are violated.

The only problem with this requirement is that a message, obviously, cannot inherit these attributes if the sender is unknown or the message is received from outside of the model. In order to solve this problem, we treat all such messages in the analysis model differently from the rest of the messages in the system. Each such message is assumed to be the initiator of the corresponding DT. The transmission type and period (in the case of periodic transmission) or inhibit time (in the case of sporadic transmission) or both (in the case of mixed transmission) [28] of such message are extracted from the user input (instead of the sending task as in the case of intra-model messages). However, the forward attribute inheritance is valid for such messages. This means that the receiver task of this message will inherit the difference between the worst- and best-case response times of the message as its release jitter.

### 9.4.6 Impact of Design Decisions in the Component Technology on the Implementation of the Analysis

The design decisions made in the component technology (i.e., RCM) can have indirect impact on the response times computed by the analysis. For exam-

ple, 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 and E2EDA, 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 a chain of components. The run-time translation of a software component may differ among different component models.

### 9.4.7 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 (if deadlines are not specified), i.e., the response times increase in every iteration.

Consider a two-node application modeled in RCM as shown in Figure 9.19 (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 the run-time allocation of network interface components (OSWC and ISWC) as shown in Figure 9.19 (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 9.19 (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. This problem may appear in those component models which do not use exclusive modeling objects or means to differentiate between intra- and inter-node communication in the design model and rely completely on the run-time environment to handle the communication. Hence, some special methods are required to avoid direct cycles in these models.

However, 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 9.19 (b). Although same messages are sent between the nodes, one task cannot

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-times, 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 may provide feedback to the design decisions concerning the run-time allocation of modeling components.

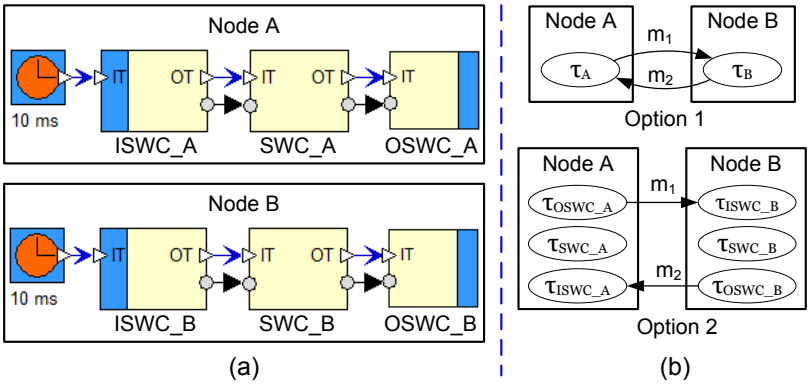


Figure 9.19: Options for the run-time allocation of network interface components

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

The plug-in framework in Rubus-ICE allows only sequential execution of plug-ins. Hence, a plug-in has to execute to completion and terminate before the next plug-in can start. It should be noted that there exists a plug-in in Rubus-ICE that can perform RTA of tasks in a node and it is already in the industrial use. There are two options to develop the HRTA plug-in for Rubus-ICE, i.e., option A and B as shown in Figure 9.20.

The option A supports reusability by building the HRTA plug-in upon the existing Node RTA Plug-in. Thus, the HRTA plug-in is built by integrating existing RTA plug-in with two new plug-ins, i.e., one implementing network RTA algorithms and the other implementing holistic RTA algorithm. In this case, the HRTA plug-in will be lightweight. 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 or the deadlines (if specified) are violated. On the other hand, option B requires the development of the 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.

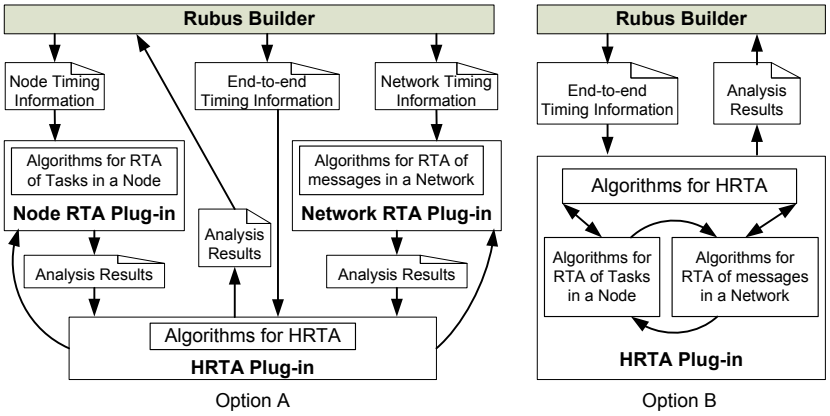


Figure 9.20: Options to develop the HRTA Plug-in for Rubus-ICE

Since, option A allows the reuse of a pre-tested and heavyweight (having most complex algorithms compared to the network and holistic RTA) node RTA plug-in, it is easy to implement and requires less time for implementation, integration and test 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 executed sequentially. Moreover, one plug-in cannot execute the other. Hence, we selected option B for the implementation of HRTA.

The algorithm for the computation of end-to-end delays requires the response times of all tasks, messages and task chains in the system as one of its inputs. As compared to HRTA algorithm, it is not iterative. Therefore, there is no need to build the E2EDA plug-in from the scratch. In fact, the HRTA plug-in can be completely reused as a black box. This means that the response times of tasks, messages and task chains computed by the HRTA plug-in can be used as one of the inputs for the E2EDA plug-in.

### 9.4.9 Analysis of DRE Systems with Multiple Networks

In a DRE system, a node may be connected to more than one network. This type of node is called a gateway node. 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 the example of a DRE system with two networks, i.e., CAN and LIN as shown in Figure 9.21. There are five nodes in the system. Node 3 is the 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 first step. In the second step, the attribute inheritance is carried out (see Section 3.3) and the subsystems are analyzed again. The second step is repeated until the response times converge or the deadlines (if specified) are violated. In the above example, we first perform HRTA using CAN and LIN networks separately. 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). Then HRTA of CAN and LIN networks is performed again. These steps are repeated until we get stable response times. Although we analyzed the subsystems separately, the multi-step analysis (especially attribute inheritance step) makes the overall analysis to be holistic.

The implemented HRTA does not support the analysis of a transaction that is distributed cyclically on multiple networks, i.e., the transactions that is distributed over more than one network while its first and last tasks are located on the same network. Since, the E2EDA plug-in receives the response-time results from the HRTA plug-in, it does not need to split the system (with multiple networks) into sub-systems. In fact, the E2EDA plug-in analyzes it as a single system.

### 9.4.10 Specification of Delay Constraints on Data Paths

One issue that concerns both modeling and analysis is how to specify the delay constraints on data paths in both data and mixed chains. This is important because the delay constraints specified in the modeled application have to be extracted in the timing model and the end-to-end delays have to be computed only for the specified data path(s) by the E2EDA plug-in. For this purpose,

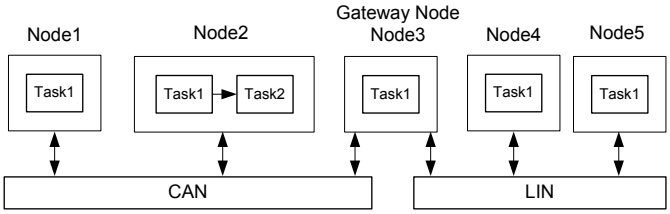


Figure 9.21: Multiple networks in a DRE system

we introduce start and end objects for each of the four delay constraints (discussed in Subsection 3.2) in the component technology. The constraint object has a meaningful name, and start and end points along a data path. Figure 9.22 shows the "Data Age" delay constraint specified on a sensor-actuator data path. Similarly, there are start and end objects for "Data Reaction", "LIFO" and "FILO" delays. All these delay constraints will be used in the case study in Section 6. In the example shown in Figure 9.22, the E2EDA plug-in will consider the tasks corresponding to the components sensor_signal_read, filter and compute_actuator_signal while calculating the data age delay. A delay constraint can also be distributed over several nodes. It should be noted that the delay constraints can be specified even on a small segment of a long data path. Another useful method for specifying the delay constraints is by selecting each component (e.g., with mouse click) along the data path. The implementation of this method in Rubus-ICE is left for the future work.

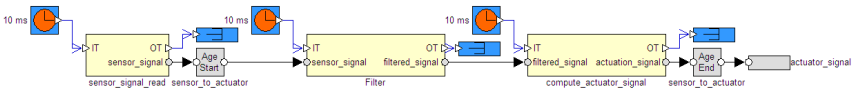


Figure 9.22: Age delay constraint specified on a data path

### 9.4.11 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 tasks and messages in the system because it may contain a lot of useless information (if the user is not interested in all of it). Furthermore, presenting

the end-to-end response times and delays 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 end-to-end response times and delays of only those tasks and DTs which have deadline requirements and delay constraints (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 6). Apart from this, we also provide an option for the user to get detailed analysis results from both the HRTA and E2EDA plug-ins. The analysis report also shows network utilization which is defined as the sum of the ratio of transmission time to the corresponding period (or minimum-update time) for all messages in the network [27].

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

We identified that it is important to provide a progress report of the HRTA and E2EDA plug-ins during their executions. 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.

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

If the analysis results indicate that the modeled system is unschedulable, it can be interesting if the 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 the HRTA plug-in will be provided in the future. Another interesting and related feature would be to provide a trace analyzer as another plug-in that can be used after system has been developed. This analyzer will record the execution of the actual system and then present a graphical comparison of the trace with response times of tasks and messages; holistic response times of trigger, data and mixed chains; and end-to-end delays of data and mixed chains. Based on such comparisons, the user may have better understanding of

how the schedulability of the system can be improved. The implementation of this feature is left for the future work.

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

Our experience of integrating the HRTA and E2EDA plug-ins with Rubus-ICE shows that there is a need for continuous collaboration between the integrator of the plug-ins and its implementer especially during the phase of integration testing (see next Section). This collaboration is more obvious when the plug-in is developed in isolation 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-ins. 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 the HRTA plug-in was non-trivial and most tedious activity.

## 9.5 Testing and Evaluation

In this section we discuss our test plan for both standalone and integration testing of the HRTA and E2EDA plug-ins. Error handling and sanity checking routines make 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 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 difference between the worst- and best-case response times of the sender task as its release jitter).

- A set of routines evaluating the overload conditions during the analysis. For example, processor or network utilization exceeding 100%, and presence of direct cycles in the system. Since HRTA algorithm is iterative, the analysis may never terminate in the presence of these conditions if the deadlines are not specified.
- A set of routines evaluating variable overflow during the analysis.
- A set of routines verifying the design correctness of the modeled application. These routines identify the presence of direct cycles in the modeled application. Moreover, they also identify if the delay constraints are wrongly specified, for example, a delay constraint specified on a trigger chain instead of a data or a mixed chain.

### 9.5.1 Standalone Testing

Standalone testing means testing of the implementation of HRTA and E2EDA before they are integrated as plug-ins with the Rubus builder tool. In other words, it refers to the testing of HRTA and E2EDA in isolation. The following input methods were used for the 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-ins corresponding to the test vectors in the first two input methods were also verified by hand.

### 9.5.2 Integration Testing

Integration testing refers to the testing of the HRTA and E2EDA plug-ins after they have been integrated with the Rubus builder tool. Although standalone testing is already performed, the integration of these plug-ins with Rubus-ICE may induce unexpected errors. Our experience shows that the integration testing is much more difficult and time consuming activity compared to the standalone testing. The following input methods were used for the integration testing.

1. Test vectors are read from external files.

2. Test vectors are manually written in the ICCM file 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-ins corresponding to all types of test cases were also verified by hand.

## 9.6 Automotive Application Case Study

We provide a proof of concept for the analysis techniques that we implemented in the Rubus-ICE tool suite by conducting the automotive-application case study. First, we model the Autonomous Cruise Control (ACC) system with RCM using Rubus-ICE. Then, we analyze the modeled ACC system using the HRTA and E2EDA plug-ins.

### 9.6.1 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 (cooperating) 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 [49].

The ACC system may be divided into four subsystems, i.e., Cruise Control, Engine Control, Brake Control and User Interface [50]. Figure 9.23 shows the block diagram of the ACC system. The subsystems communicate with each other via the CAN network.

**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 that corresponds to ON/OFF, Standby and Resume (resuming to a speed predefined by the driver) 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.

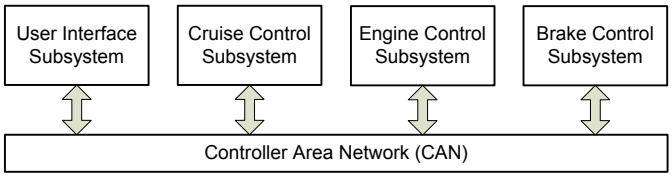


Figure 9.23: Block diagram of Autonomous Cruise Control System

**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 the 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 the 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 the 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 the CC subsystem. Based on this feedback, it computes new vehicle speed. Accordingly, it produces control signals and sends them to the brake actuators and brake light controllers. It also sends CAN messages to other subsystems that carry information regarding status of manual brake, vehicle speed and RPM.

### 9.6.2 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 9.24. The selected speed of the CAN bus is 500 kbps. The extended frame format is selected which means that each frame will use 29-bit identifier [51]. The ACC system is modeled with trigger, data and mixed chains.

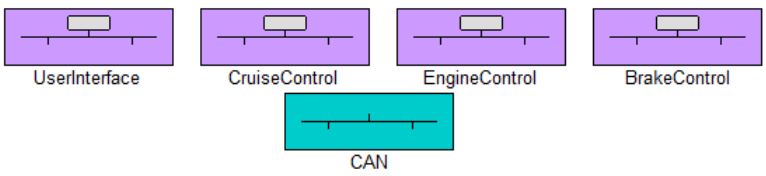


Figure 9.24: Autonomous Cruise Control System modeled with RCM

Table 9.1: Message attributes extracted from the model

| **Msg** | $s_m$ | $P_m$ | $\xi_m$ | $T_m$ ($\mu$**Sec**) | $C_m$ ($\mu$**Sec**) |
|---|---|---|---|---|---|
| $m1$ | 8 | 7 | Periodic | 10000 | 320 |
| $m2$ | 8 | 6 | Periodic | 10000 | 320 |
| $m3$ | 8 | 4 | Periodic | 10000 | 200 |
| $m4$ | 8 | 3 | Sporadic | 10000 | 320 |
| $m5$ | 2 | 5 | Sporadic | 10000 | 320 |
| $m6$ | 2 | 2 | Periodic | 10000 | 200 |
| $m7$ | 1 | 1 | Sporadic | 10000 | 180 |

There are seven CAN messages that are sent by the nodes as shown in Figure 9.25. A signal data base "signalDB" 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 9.1.

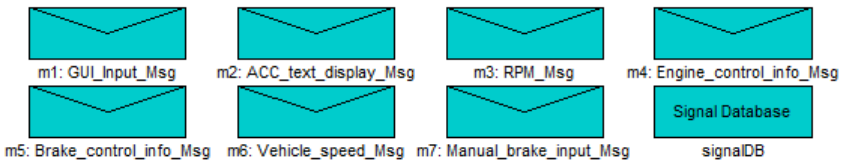


Figure 9.25: CAN messages and signal database modeled with RCM

The high-level architectures of CC, EC, BC and UI nodes modeled with RCM are shown in Figure 9.26, 9.27, 9.28 and 9.29 respectively.

**Internal Model of Cruise Control Node in RCM.**

The CC node is modeled with four assemblies as shown in Figure 9.26. An assembly in RCM is a container for various software items. The Input_from_Sensors assembly contains one SWC that reads radar sensor values as shown in Figure 9.30. 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 9.31. These components receive messages $m1$,

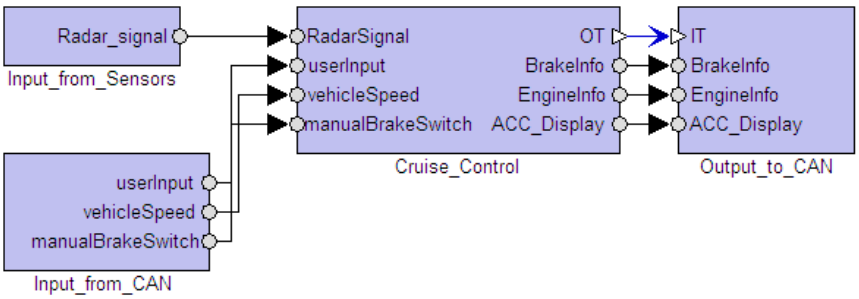


Figure 9.26: RCM model of the Cruise Control node

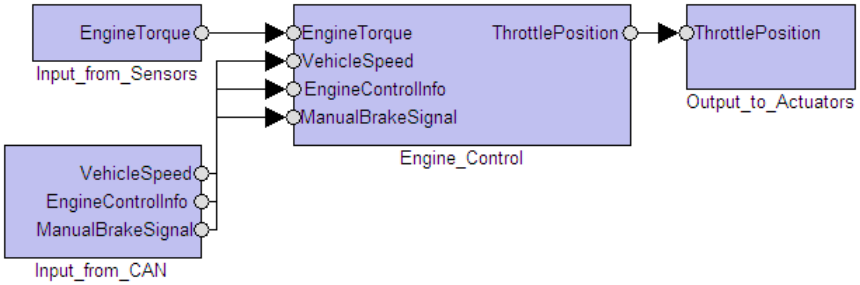


Figure 9.27: RCM model of the Engine Control node

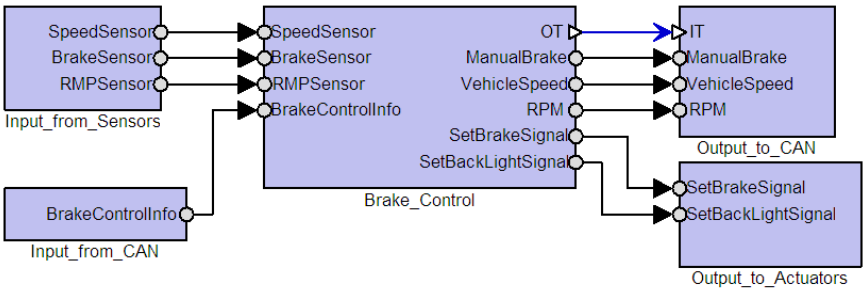


Figure 9.28: RCM model of the Brake Control node

$m6$ and $m7$ from the CAN network respectively. Similarly, the assembly Output_to_CAN contains three OSWC components as shown in Figure 9.32. These

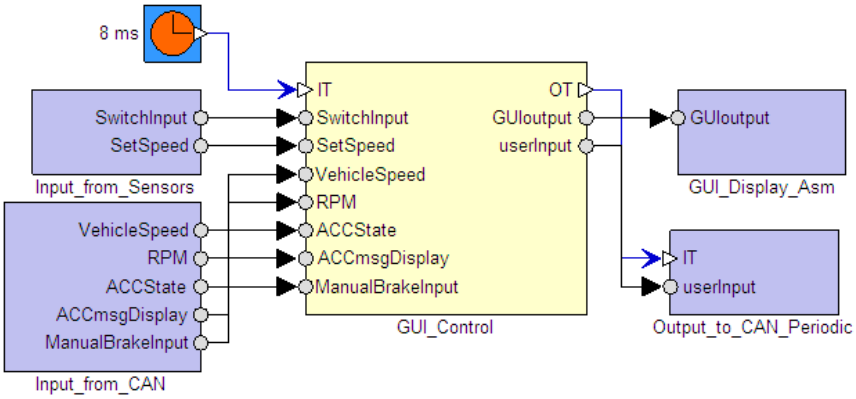


Figure 9.29: RCM model of the User Interface node

components send messages $m5$, $m4$ and $m2$ to the 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 messages for the other nodes. The internal model of this assembly is shown in Figure 9.33.

**Internal Model of Engine Control Node in RCM.**

The Engine Control node is modeled with four assemblies as shown in Figure 9.27. The Input_from_Sensors assembly contains one SWC that reads the sensor values corresponding to the engine torque as shown in Figure 9.34. 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 9.35. These components receive messages $m6$, $m4$ and $m7$ from the CAN network respectively. The third assembly, Output_to_Actuators as shown in Figure 9.36, contains the SWC that produces control signals for the engine throttle actuator. The fourth assembly, i.e., Engine_Control as shown in Figure 9.37, contains two SWCs: one handles and processes the inputs from sensors and received messages, while the other computes the new position for the engine throttle. These components are part of a distributed mixed chain that we will analyze along with other distributed mixed chains in the next subsections.

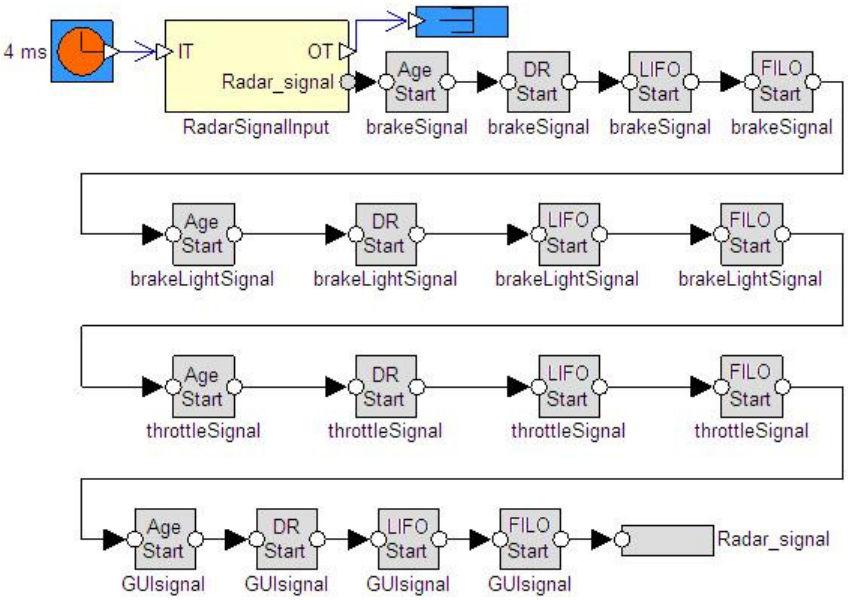


Figure 9.30: CC node: Internal model of the Input_from_Sensors assembly

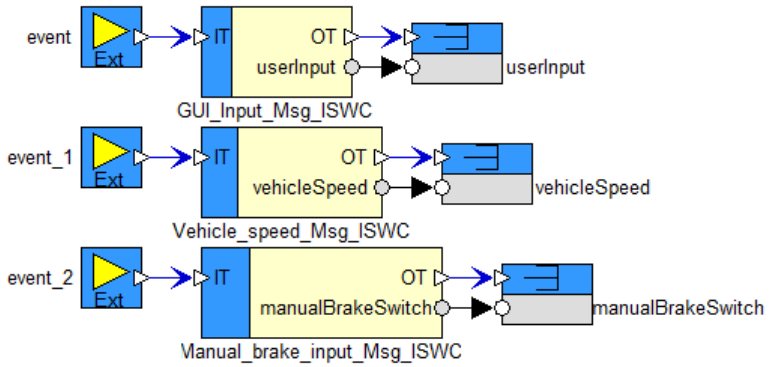


Figure 9.31: CC node: Internal model of the Input_from_CAN assembly

**Internal Model of Brake Control Node in RCM.**

The Brake Control node is modeled with five assemblies as shown in Figure 9.28. The Input_from_Sensors assembly contains three SWCs as shown in Fig-

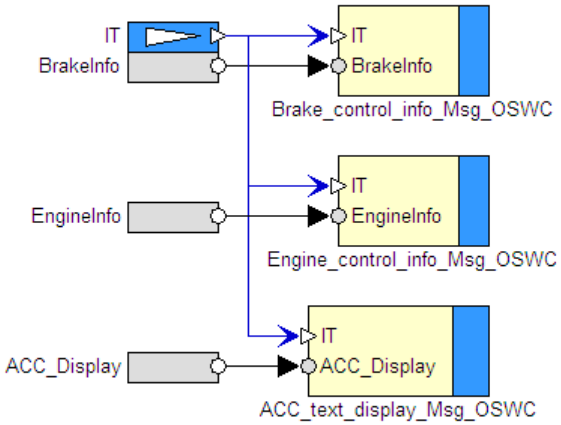


Figure 9.32: CC node: Internal model of the Output_to_CAN assembly

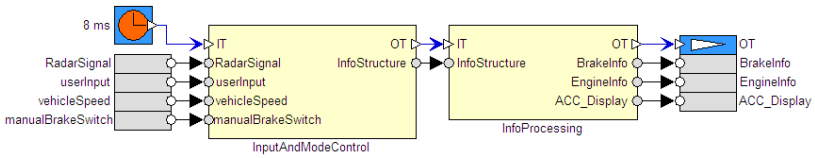


Figure 9.33: CC node: SWCs comprising the Cruise_Control assembly

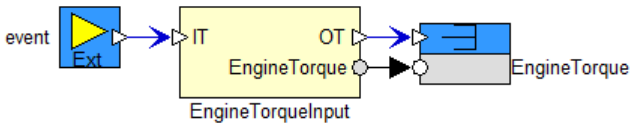


Figure 9.34: EC node: Internal model of the Input_from_Sensors assembly

ure 9.38. 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 assembly, shown in Figure 9.39, contains the ISWC component Brake_control_info_Msg_ISWC that receives a message $m5$ from the CAN network.

The third assembly, i.e., Brake_Control as shown in Figure 9.40, 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

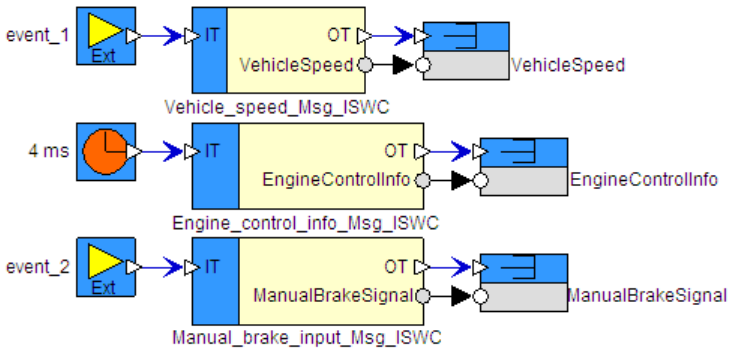


Figure 9.35: EC node: Internal model of the Input_from_CAN assembly

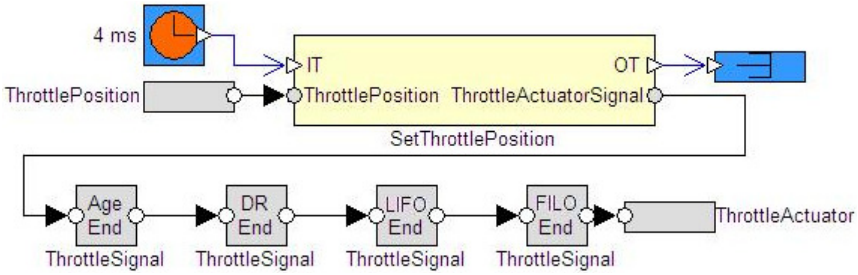


Figure 9.36: EC node: Internal model of the Output_to_Actuators assembly

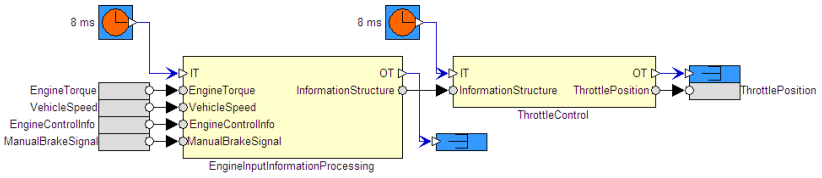


Figure 9.37: EC node: SWCs comprising the Engine_Control assembly

fourth assembly Output_to_CAN contains three OSWC components as shown in Figure 9.41. These components send messages $m7$, $m6$ and $m3$ to the CAN network. The fifth assembly, Output_to_Actuators as shown in Figure 9.42, contains the SWCs that produce control signals for the brake actuators

and brake light controllers.

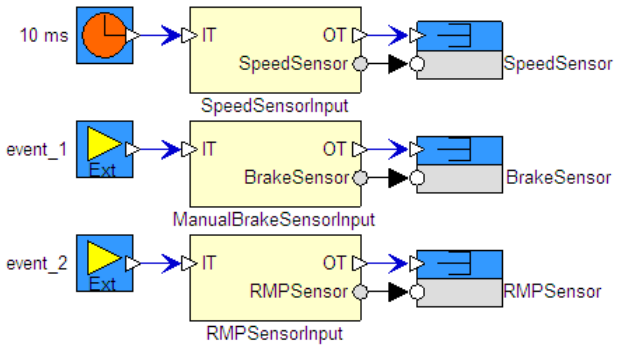


Figure 9.38: BC node: Internal model of the Input_from_Sensors assembly

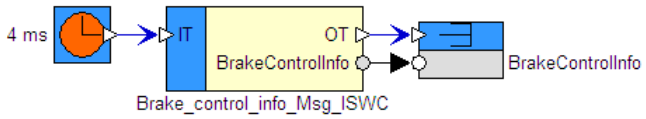


Figure 9.39: BC node: Internal model of the Input_from_CAN assembly

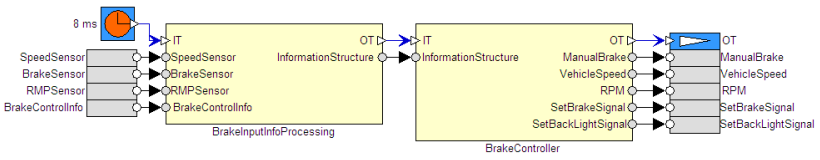


Figure 9.40: BC node: Internal model of the Brake_Control assembly

**Internal Model of User Interface Node in RCM.**

The User Interface node is modeled with four assemblies along with one SWC as shown in Figure 9.29. 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

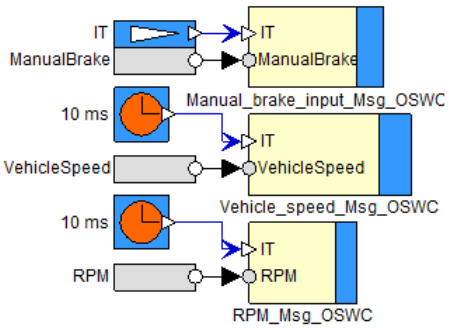


Figure 9.41: BC node: Internal model of the Output_to_CAN assembly

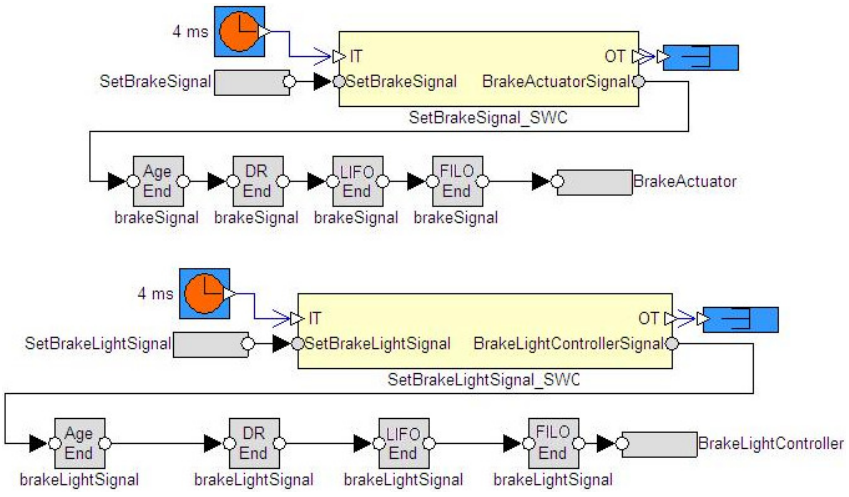


Figure 9.42: BC node: Internal model of the Output_to_Actuators assembly

also computes control signals for the other nodes. The Input_from_Sensors assembly contains two SWCs as shown in Figure 9.43. 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. The Input_from_CAN assembly contains four ISWC components, i.e., Vehicle_Speed_Msg_ISWC, RPM_Msg_ISWC, Manual_brake_input_Msg_ISWC and ACC_text_display_Msg_ISWC as

shown in Figure 9.44. These components receive messages $m6$, $m3$, $m7$ and $m2$ from the CAN network respectively. The third assembly, i.e., Output_to_CAN_Periodic sends a message $m1$ to the CAN network via the OSWC component as shown in Figure 9.45. The fourth assembly, i.e., GUI_Display _Asm contains one SWC, i.e., GUIdisplay component as shown in Figure 9.46. This component sends the signals (corresponding to updated information) to GUI in the car.

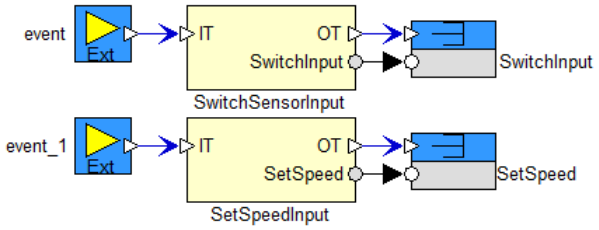


Figure 9.43: UI node: Internal model of the Input_from_Sensors assembly

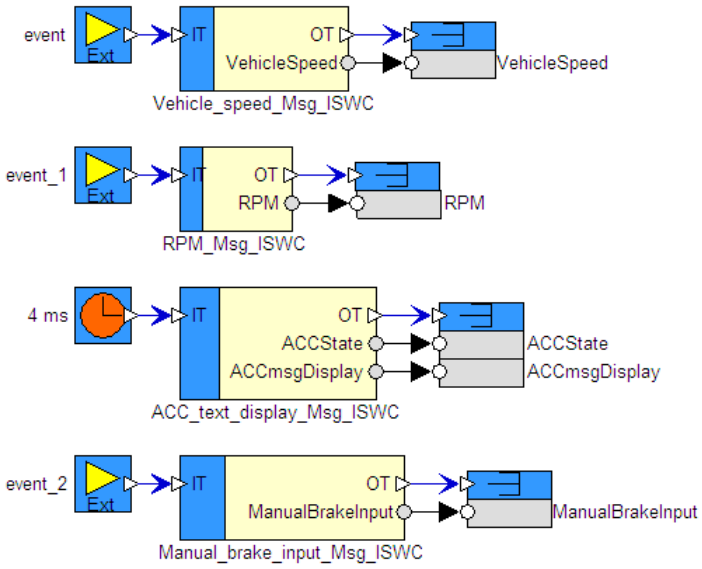


Figure 9.44: UI node: Internal model of the Input_from_CAN assembly

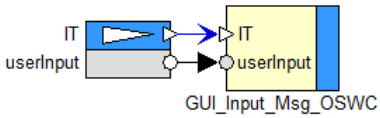


Figure 9.45: UI node: Internal model of the Output_to_CAN_Periodic assembly

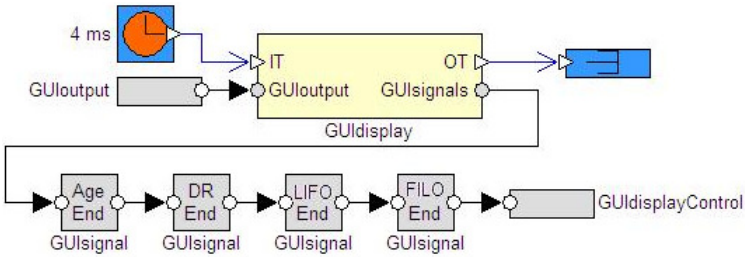


Figure 9.46: UI node: Internal model of the GUI_Display_Asm assembly

### 9.6.3 Modeling of End-to-end Deadline Requirements

We specify end-to-end deadline requirements on four DTs in the ACC system using a deadline object in RCM. All these DTs, i.e., $DT_1$, $DT_2$, $DT_3$ and $DT_4$ are distributed mixed chains as shown in Table 9.2. All these chains have one common initiator, i.e., their first task corresponds to the SWC that reads radar signal which is denoted by *RadarSignalInput* and located in the Cruise Control node as shown in Figure 9.30. The last tasks of $DT_1$ and $DT_2$ are located in the Brake Control node. These tasks correspond to the SWCs *SetBrakeSignal* and *SetBrakeLightSignal* as shown in Figure 9.28. These two tasks are responsible for producing brake actuation and brake light control signals respectively. The last task of $DT_3$ corresponds to *SetThrottlePosition* SWC and is located in the Engine Control node as shown in Figure 9.27. This task is responsible for producing control signal for the engine throttle actuator. The last task of $DT_4$ corresponds to *GUIdisplay* SWC and is located in the User Interface node as shown in Figure 9.29. This task is responsible for providing display information for the driver.

All the mixed chains under analysis are distributed over more than one node. For the sake of convenience, we list all the components in the data path (from initiator to terminator) of each chain as shown below. We also specify four delay constraints (discussed in Section 3) on each DT under analysis. In

RCM, the model of each delay constraint consists of start object and end object. The start objects for all four delay constraints for each DT are shown in Figure 9.30. There are sixteen start objects for delay constraints in Figure 9.30 because there are four DTs under analysis with four delay constraints specified on each DT. The end objects for all delay constraints for $DT_1$ and $DT_2$ are specified in Figure 9.42. Similarly, the end objects for all delay constraints for $DT_3$ and $DT_4$ are specified in Figure 9.36 and Figure 9.46 respectively.

1. $DT_1$: $RadarSignalInput \rightarrow InputAndModeControl \rightarrow InfoProcessing \rightarrow Brake\_control\_info\_Msg\_OSWC \rightarrow message : m5 \rightarrow Brake\_control\_info\_Msg\_ISWC \rightarrow BrakeInputInfoProcessing \rightarrow BrakeController \rightarrow SetBrakeSignal\_SWC$

2. $DT_2$: $RadarSignalInput \rightarrow InputAndModeControl \rightarrow InfoProcessing \rightarrow Brake\_control\_info\_Msg\_OSWC \rightarrow message : m5 \rightarrow Brake\_control\_info\_Msg\_ISWC \rightarrow BrakeInputInfoProcessing \rightarrow BrakeController \rightarrow SetBrakeLightSignal\_SWC$

3. $DT_3$: $RadarSignalInput \rightarrow InputAndModeControl \rightarrow InfoProcessing \rightarrow Engine\_control\_info\_Msg\_OSWC \rightarrow message : m4 \rightarrow Engine\_control\_info\_Msg\_ISWC \rightarrow EngineInputInformationProcessing \rightarrow ThrottleControl \rightarrow SetThrottlePosition$

4. $DT_4$: $RadarSignalInput \rightarrow InputAndModeControl \rightarrow InfoProcessing \rightarrow ACC\_text\_display\_Msg\_OSWC \rightarrow message : m2 \rightarrow ACC\_text\_display\_Msg\_ISWC \rightarrow GUI\_Control \rightarrow GUIdisplay$

### 9.6.4 Analysis of ACC System using the HRTA and E2EDA Plug-ins

The run-time allocation of all the components in the model of the ACC system results in 19 transactions, 36 tasks and 7 messages. We provide the analysis results of only those transactions on which deadline requirements or delay constraints are specified. The transmission times ($C_m$) of all messages computed by the HRTA plug-in are shown in Table 9.1. The WCET of each component in the modeled ACC system is selected from the range of 10-60 $\mu$Sec. The HRTA

Table 9.2: Analysis report by the HRTA plug-in

| **Distributed Transaction** | **Chain Type** | **Control Signal Produced by the Chain** | **Deadline ($\mu$Sec)** | **Holistic Response Time ($\mu$Sec)** |
|---|---|---|---|---|
| $DT_1$ | Mixed | SetBrakeSignal | 1000 | 220 |
| $DT_2$ | Mixed | SetBrakeLightSignal | 1000 | 280 |
| $DT_3$ | Mixed | SetThrottlePosition | 1000 | 130 |
| $DT_4$ | Mixed | GUIdisplay | 1500 | 345 |

plug-in analyzes all four DTs (discussed in the previous subsection). Once the HRTA plug-in has completed its execution and produced analysis results then the E2EDA plug-in analyzes only those DTs on which end-to-end delay constraints are specified (i.e., all four DTs).

The analysis report in Table 9.2 provides worst-case holistic response times of the four distributed mixed chains using the 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 times of these four DTs correspond to the production of control signals for brake actuators, brake lights controllers, engine throttle actuator and graphical user interface.

The analysis report produced by the E2EDA plug-in is shown in Table 9.3. It lists four end-to-end delays calculated for each DT under analysis. The corresponding specified delay constraints are also listed in the table. By comparing the end-to-end deadlines and specified delay constraints with the calculated holistic response times and end-to-end delays in Tables 9.2 and 9.3 respectively, we see that the modeled ACC system meets all of its deadlines.

## 9.7 Conclusion and Future Work

We presented the implementation of the state-of-the-art Holistic Response Time Analysis (HRTA) and End-to-End Delay Analysis (E2EDA) as two individual plug-ins for the existing industrial tool suite Rubus-ICE. The implemented analyses are general as they support the integration of real-time analysis of various networks without a need for changing the end-to-end analysis algorithms.

Table 9.3: Analysis report by the E2EDA plug-in

| **Distributed Transaction** | $DT_1$ | $DT_2$ | $DT_3$ | $DT_4$ |
|---|---|---|---|---|
| **Specified Age Delay** | 5000 | 5000 | 5000 | 5000 |
| **Constraint($\mu$Sec)** | | | | |
| **Calculated Age Delay ($\mu$Sec)** | 4220 | 4280 | 4130 | 4345 |
| **Specified Reaction Delay** | 10000 | 10000 | 10000 | 10000 |
| **Constraint($\mu$Sec)** | | | | |
| **Calculated Reaction Delay ($\mu$Sec)** | 8220 | 8280 | 8130 | 8345 |
| **Specified LIFO Delay** | 1000 | 1000 | 1000 | 1500 |
| **Constraint($\mu$Sec)** | | | | |
| **Calculated LIFO Delay ($\mu$Sec)** | 220 | 280 | 130 | 345 |
| **Specified FILO Delay** | 15000 | 15000 | 15000 | 15000 |
| **Constraint($\mu$Sec)** | | | | |
| **Calculated FILO Delay ($\mu$Sec)** | 12220 | 12280 | 12130 | 12345 |

With the implementation of these plug-ins, Rubus-ICE is able to support distributed end-to-end timing analysis of trigger flows as well as asynchronous data flows which are common in automotive embedded systems.

There are many challenges faced by the implementer when state-of-the-art real-time analyses like HRTA and E2EDA are transferred to the industrial tools. The implementer has to not only code and implement the analyses in the tools, but also deal with various challenging issues in an effective way with respect to time and cost. We discussed and solved several issues that we faced during the implementation, integration and evaluation of the HRTA and E2EDA plug-ins. The experience gained by dealing with the implementation challenges provided a feed back to the component technology (i.e., the Rubus 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 the HRTA and E2EDA plug-ins. We found the integration testing to be a tedious and non-trivial activity. Our experience of implementing, integrating and evaluating these plug-ins 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 the autonomous cruise control system with component-based development approach using the existing indus-

trial component model (Rubus Component Model) and analyzing it with the HRTA and E2EDA plug-ins.

We believe that most of the problems 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 allows component-based development of distributed real-time embedded 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 the HRTA plug-in. Another future work is the implementation of RTA for CAN with FIFO and work-conserving queues [26, 25], and RTA of CAN with FIFO Queues for Mixed Messages [28] within HRTA plug-in. We also plan to integrate the stand alone analyzer, that we developed for the analysis of mixed messages with offsets [30], with the HRTA plug-in.

# Acknowledgement

This work is supported by the 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, BAE Systems Hägglunds and Volvo Construction Equipment (VCE) Sweden.

# Appendix A

## Acronyms and Abbreviations

| | |
|---|---|
| **ACC** | Autonomous Cruise Control |
| **API** | Application Programming Interface |
| **AUTOSAR** | AUTomotive Open System ARchitecture |
| **BC** | Brake Control |
| **BCRT** | Best Case Response Time |
| **CAN** | Controller Area Network |
| **CC** | Cruise Control |
| **DR** | Data Reaction |
| **DRE** | Distributed Real-time Embedded |
| **DT** | Distributed Transaction |
| **EC** | Engine Control |
| **E2EDA** | End To End Delay Analysis |
| **FIFO** | First In First Out |
| **FILO** | First In Last Out |
| **HCAN** | Hägglunds Controller Area Network |
| **HRTA** | Holistic Response Time Analysis |
| **ICCM** | Intermediate Compiled Component Model |
| **ICE** | Integrated Component development Environment |
| **ISWC** | Input Software Circuit |
| **LIFO** | Last In First Out |
| **LILO** | Last In Last Out |
| **OSWC** | Output Software Circuit |
| **RCM** | Rubus Component Model |
| **RTA** | Response Time Analysis |
| **SWC** | Software Circuit |
| **TIMMO** | TIMing MOdel |
| **TP** | Timed Path |
| **TTCAN** | Time Triggered Controller Area Network |
| **UI** | User Interface |
| **WCET** | Worst Case Execution Time |
| **WCRT** | Worst Case Response Time |

# 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. *IEEE International Symposium on Real-Time Systems*, page 26, 1998.

[6] 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 *Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS)*, December 2008.

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

[8] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Support for Holistic Response-time Analysis in an Industrial Tool Suite: Implementation

Issues, Experiences and a Case Study. In *19th IEEE Conference on Engineering of Computer Based Systems (ECBS)*, pages 210 –221, April 2012.

[9] BAE Systems Hägglunds. http://www.baesystems.com/hagglunds.

[10] Volvo Construction Equipment. http://www.volvoce.com.

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

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

[13] 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.

[14] Saad Mubeen. Modeling and timing analysis of industrial component-based distributed real-time embedded systems. Licentiate thesis, Mälardalen University, January 2012.

[15] 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)*, pages 229–238, September 2011.

[16] 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.

[17] K. Hänninen et.al. Framework for real-time analysis in Rubus-ICE. In *13th IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 782 –788, 2008.

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

[19] 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.

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

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

[22] Friedhelm Stappert, Jan Jonsson, Jürgen Mottok, and Rolf Johansson. A Design Framework for End-To-End Timing Constrained Automotive Applications. In *Embedded Real-Time Software and Systems (ERTS), 2010*.

[23] 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.

[24] 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.

[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*, July 2011.

[26] R. Davis and N. Navet. Controller Area Network (CAN) Schedulability Analysis for Messages with Arbitrary Deadlines in FIFO and Work-Conserving Queues. In *9th IEEE International Workshop on Factory Communication Systems (WFCS)*, pages 33 –42, May 2012.

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

[28] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Response-Time Analysis of Mixed Messages in Controller Area Network with Priority- and FIFO-Queued Nodes. In *9th IEEE International Workshop on Factory Communication Systems (WFCS)*, May 2012.

[29] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Extending response-time analysis of controller area network (CAN) with FIFO

queues for mixed messages. In *16th IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–4, September 2011.

[30] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Worst-case response-time analysis for mixed messages with offsets in controller area network. In *17th IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, September 2012.

[31] 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, USA, 2002. ACM.

[32] 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.

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

[34] A. C. Rajeev, Swarup Mohalik, Manoj G. Dixit, Devesh B. Chokshi, and S. Ramesh. Schedulability and end-to-end latency in distributed ecu networks: formal modeling and precise estimation. In *Proceedings of the tenth ACM international conference on Embedded software*, EMSOFT '10, pages 129–138. ACM, 2010.

[35] MAST–Modeling and Analysis Suite for Real-Time Applications. http://mast.uni-can.es.

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

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

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

[39] M. Hagner and U. Goltz. Integration of scheduling analysis into uml based development processes through model transformation. In *International Multi-conference on Computer Science and Information Technology (IMCSIT)*, pages 797 –804, October 2010.

[40] Vector. http://www.vector.com.

[41] CANoe. http://www.vector.com/portal/medien/cmc/info/canoe_product-information_ en.pdf.

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

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

[44] M. Hagner and U. Goltz. Integration of scheduling analysis into uml based development processes through model transformation. In *International Multi-conference on Computer Science and Information Technology (IMCSIT)*, pages 797 –804, October 2010.

[45] CANopen Application Layer and Communication Profile. CiA Draft Standard 301. Version 4.02. February 13, 2002. http://www.can-cia.org/-index.php?id=440.

[46] Requirements on Communication, Release 3.0, Revision 7, Ver. 2.2.0. The AUTOSAR Consortium, September, 2010. www.autosar.org/download/R3.0/AUTO-SAR_SRS_COM.pdf.

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

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

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

[50] 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.

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

# Chapter 10

# Paper G: Component-Based Vehicular Distributed Embedded Systems: End-to-end Timing Models Extraction at Various Abstraction Levels

Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin
MRTC report ISSN 1404-3041, Mälardalen Real-Time Research Centre, Mälardalen University, May, 2014 (pending submission as conference contribution).

**Abstract**

In order to perform the end-to-end response-time and delay analyses of a system, its end-to-end timing model should be available. The majority of existing model- and component-based development approaches for vehicular distributed embedded systems extract the end-to-end timing model at an abstraction level and development phase that is close to the system implementation. We present a method to extract the end-to-end timing models from the systems at a higher abstraction level. At the higher level, the method extracts timing information from system models that are developed with EAST-ADL and Timing Augmented Description Language (TADL2) using the TIMMO methodology. At the lower level, the method exploits the Rubus component model to extract the timing information that cannot be clearly specified at the higher level such as trigger paths in distributed chains. We also discuss challenges and issues faced during extraction of the timing models. Further, we present guidelines and solutions to address these challenges.

## 10.1 Introduction

Due to increase in the amount of advanced computer controlled functionality in vehicular distributed embedded systems, the size and complexity of embedded software has drastically increased in the past few years. For example, the embedded software in heavy vehicle architectures such as a truck may consist of as many as 2000 software functions that may be distributed over 45 ECUs (Electronic Control Units) [1]. In order to deal with the software complexity, the research community proposed model- and component-based development of embedded real-time systems by using the principles of Model-Based Software Engineering (MBSE) and Component-Based Software Engineering (CBSE) [2, 3]. This approach is intended to capture requirements early during the development, lower development cost, enable faster turn-around times in early design phases, increase reusability, support modeling at higher abstraction levels, and to provide possibilities to automatically perform timing analysis; derive test cases; and generate code. MBSE provides the means to use models to describe functions, structures and other design artifacts. Whereas, CBSE supports the development of large software systems by integration of software components. It raises the level of abstraction for software development and aims to reuse software components and their architectures. Within the segment of construction-equipment vehicles and similar segments for heavy special-purpose vehicles, model-based development of software architectures for embedded real-time systems has had a surge the last few years.

### 10.1.1 Problem Statement and Paper Contributions

Most of the vehicular functions are developed as distributed embedded systems with real-time requirements. Hence, the providers of these systems are required to ensure that the actions by the systems will be taken at a time that is appropriate to their environment. One way to guarantee that the system will meet all its deadlines is to perform the end-to-end response-time and delay analyses [4, 5]. These analyses can validate timing requirements, specified on the system, without performing exhaustive testing. In order to perform the timing analyses of the system, an end-to-end timing model needs to be available. The end-to-end timing model consists of the information containing timing properties, requirements and dependencies concerning all tasks, messages, task chains and distributed transactions in the system. Based on this information, execution behavior of the system, with respect to timing, can be predicted by means of certain type of timing analysis such as end-to-end response-time

and delay analysis.

The majority of existing approaches for component-based vehicular distributed embedded systems support the extraction of the timing models at an abstraction level and development phase that is close to their implementation [6, 7, 8, 5, 9, 10]. An abstraction level provides a complete definition of the system for a given purpose during the development process. Furthermore, high-level timing analysis provided by existing approaches does not support high precision and is often not based on actual implementation of the system. As a result, a high-precision end-to-end timing analysis cannot be performed at higher abstraction levels. In the past few years, one of the focuses of several large EU research projects, that involve both academia and industry, has been on supporting the timing analysis at various abstraction levels and development phases [11, 12, 13].

Extraction of the timing model at higher abstraction levels is challenging mainly because not all timing information that is required to be captured in the timing model is available. Moreover, mismatch and incompatibilities among various methodologies, languages and tools that are used in different development phases also add to the complexity of extracting the timing model. Since complete timing information may not be available at higher levels, the timing analysis results may not represent accurate timing behavior of the final system. However, these results can provide useful information that can guide further model refinement and implementation.

We envision the extraction of end-to-end timing model and performing high-precision end-to-end timing analysis at higher levels of abstraction to be state of the practice in the future. We believe, timing information will be formally modeled at higher abstraction levels in the vehicular industry. In that case, we need to extract the specified timing information at higher abstraction levels and connect it to the implementation to generate the end-to-end timing model. Otherwise, it can be too late to extract the timing model at lower abstraction levels that are close to system implementation.

We have experienced that timing information is modeled at higher abstraction levels in the vehicular industry. This may be carried out using SysML language [14]. However, it is done mostly in an informal and textual way; which cannot be used for any formal timing analysis. Today, Timing Augmented Description Language (TADL2) [15] provides the only viable formal method for modeling of timing information using timing constraints at various abstraction levels. In order to extract a complete end-to-end timing model and perform a high-precision timing analysis, TADL2 has to be combined with a lower abstraction level execution modeling approach such as the Rubus Com-

ponent Model (RCM) [16]. We hope the industry will start using TADL2. If they do so, we can reuse that information to perform high-precision end-to-end response-time and delay analyses at a higher abstraction level.

In our previous work [8], we presented a method to extract end-to-end timing models at lower abstraction levels. In this paper, we extend our previous method to extract the timing models at a higher abstraction level. At the higher level, the method extracts timing information from system models that are developed with EAST-ADL [17] using the TIMMO methodology [18]; and annotated with timing information using TADL2. At the lower level, the method exploits RCM and its tool suite Rubus-ICE [19] to extract the timing information that cannot be clearly specified at the higher level, e.g., trigger paths in distributed chains. However, it is not straightforward to combine TADL2 with RCM due to several challenges such as unambiguous transformation of TADL2 timing constraints in RCM; and unambiguous extraction of control and data paths at the higher level. The main focus of this paper is to attack these challenges. As a contribution, we provide an interpretation of TADL2 timing constraints in RCM. Moreover, we propose extensions in RCM for unambiguous transformation of TADL2 timing constraints.

We choose RCM instead of AUTOSAR [6] at the lower abstraction level. This is because AUTOSAR lacks a complete timing model, e.g., control flow is not specified in an unambiguous way. The work in this paper is a step towards a bigger goal, i.e., development of a seamless tool-chain for model-based development of vehicular distributed real-time systems; and support for interoperating various modeling and analysis tools, including the AUTOSAR-based tool chain [13].

### 10.1.2 Paper Layout

The rest of the paper is organized as follows. In Section 10.2, we discuss background and related work. In Section 10.3, we discuss an interpretation of TADL2 timing constraints in RCM. In Section 10.4 we discuss other challenges and corresponding solutions. Finally, Section 10.5 summarizes the paper and presents the future work.

## 10.2 Background and Related Works

### 10.2.1 Abstraction Levels Considered by Various Methodologies

There are several frameworks that support timing modeling such as AADL [20], SCADE [21], MARTE [22], MAST [23], SysML, CHESS [24, 25]. However, the focus in the vehicle industry, especially in the segment of construction-equipment vehicles and other heavy road vehicle architectures, today is on EAST-ADL and AUTOSAR; Rubus is also being used. In this work, we focus only on the vehicular domain. Various models and methodologies used for the development of vehicular distributed embedded systems [17, 15, 6, 26] consider four *abstraction levels* shown in Figure 10.1.

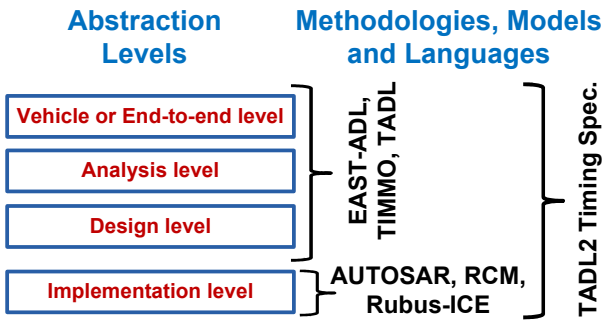


Figure 10.1: Abstraction levels considered during the development

#### Vehicle or end-to-end level

At the vehicle level, requirements, functionality and features of the vehicle are captured in an informal (often textual) and solution-independent way. This level captures the information regarding what the system should do [27]. In the segment of construction-equipment vehicles, this abstraction level is better known as the end-to-end level because features and requirements on the end-to-end functionality of the machine or vehicle are captured in an informal way.

#### Analysis level

At the analysis level, the requirements are formally captured in a allocation-independent way. Functionality of the system is defined based on the requirements and features without implementation details. A high-level analysis may also be performed for functional verification.

#### Design level

The artifacts at this level are developed in an implementation-independent way. These artifacts are refined from the analysis level artifacts; and also contains middleware abstraction and hardware architecture. In addition, software functions to hardware allocation may be present.

#### Implementation level

At the implementation level, the design-level artifact is refined to software-based implementation of the system functionality. The EAST-ADL methodology defines a system at this level in terms of AUTOSAR elements. However, in this work, our focus is on using RCM and its development environment Rubus-ICE at the implementation level. Hence, the artifact at this level consists of software architecture of the system defined in terms of Rubus components and their interactions.

In this work, we focus on the extraction end-to-end timing models mainly at the design and implementation levels.

### 10.2.2 Models and Development Methodologies

We focus on some of the component technologies that are used for the development of distributed embedded systems in the vehicular domain.

#### Rubus Component Model (RCM) and Rubus-ICE

Rubus [28] is a collection of methods and tools for model- and component-based development of dependable embedded real-time systems. It is developed by Arcticus Systems [1] in close collaboration with several industrial partners. Rubus is today mainly used for the development of control functionality in vehicles by several international companies, e.g., BAE Systems Hägglunds[2],

[1] http://www.arcticus-systems.com
[2] http://www.baesystems.com/hagglunds

Volvo Construction Equipment[3], Knorr-bremse[4], Mecel[5] and Hoerbiger[6]. The Rubus concept is based around RCM 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, timing analyzable and synthesizable control functions in resource-constrained embedded systems. The timing analysis supported by Rubus-ICE includes distributed end-to-end response-time and delay analyses [5]. Rubus methods and tools mostly focus at the implementation level in Figure 10.1.

The lowest-level hierarchical component in RCM is called Software Circuit (SWC). Its purpose is to encapsulate basic functions. Figure 10.2 shows an example of a software architecture in RCM composed of SWCs; interconnections between SWCs; and their interactions with external events and actuators with regard to both data and triggering.

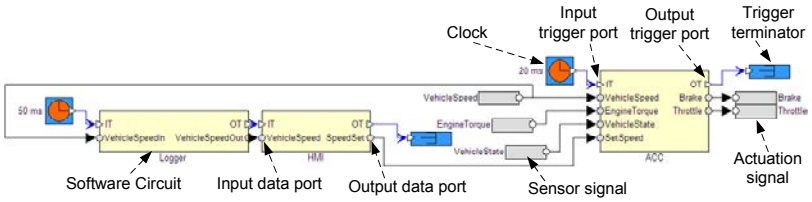


Figure 10.2: Example of software architecture of a system modeled in RCM.

## AUTOSAR

AUTOSAR [29] is an industrial initiative to provide standardized software architecture for the development of embedded software. It is used at the implementation level in Figure 10.1. It describes software development at a higher level of abstraction compared to RCM. Unlike RCM, it does not separate control and data flows among components within a node. It does not differentiate between the modeling of intra- and inter-node communication which is unlike RCM. The timing model in AUTOSAR is introduced fairly recently compared to that of Rubus. There are some similarities between AUTOSAR and RCM,

[3]http://www.volvoce.com
[4]http://www.knorr-bremse.com
[5]http://www.mecel.se
[6]http://www.hoerbiger.com

e.g., the sender receiver communication in AUTOSAR resembles the pipe-and-filter communication in RCM. AUTOSAR is more focussed on the functional and structural abstractions, hiding the implementation details about execution and communication. AUTOSAR hides the details that RCM highlights.

### EAST-ADL, MARTE, TIMMO, TIMMO-2-USE, TADL and TADL2

TIMMO [11] is an initiative to provide AUTOSAR with a timing model [30]. It is based around a methodology and TADL [26] language. TADL is used to express timing requirements and constraints. It is inspired by MARTE [22] which is a UML profile for model-driven development of real-time and embedded systems. TIMMO methodology uses EAST-ADL language [17] for structural modeling and AUTOSAR for the implementation. TIMMO and EAST-ADL focus on the top three levels in Figure 10.1. However, TIMMO methodology uses AUTOSAR at the implementation level.

In TIMMO-2-USE project [12], a major redefinition of TADL is done and released in TADL2 specification [15]. TADL2 can specify timing related information at all abstraction levels shown in Figure 10.1. Most of these initiatives lack the support on expressing low-level details at the higher levels such as linking information in distributed chains. These details are necessary to extract the end-to-end timing model from the architecture. Furthermore, there is no support on how to extract this information from the model or perform timing analysis. In our view, the end-to-end timing model includes enough information from the systems to be able to perform certain type of timing analysis, e.g., end-to-end response-time analysis.

To the best of our knowledge, none of the existing approaches are able to extract the end-to-end timing model and perform corresponding pre run-time analysis at higher levels of abstraction. This is one of the objectives in an ongoing EU research project [13].

### COMDES and ProCom

COMDES-II [9] is a two-level component-based framework for the development of distributed embedded control systems. Unlike RCM, COMDES-II employs signal-based communication for both intra- and inter-node interactions. COMDES-II does not include explicit components to model network communication.

ProCom [7] is a two-layered component model for the development of distributed embedded systems. It is inspired by RCM and there are a number

of similarities between the two. For example, both have passive components, both separate control flow from the data flow, and both use a pipe-and-filter style of communication mechanism for components interconnection. However, ProCom does not differentiate between intra- and inter-node communication which is unlike RCM. ProCom hides communication details, whereas RCM lifts them up to the modeling level.

ProCom supports timing analysis [31], however, the analysis is not performed with such a high precision as it is done in Rubus-ICE. Moreover, the timing analysis in ProCom does not include execution models. It will be very hard in ProCom and COMDES-II to extract the timing model and perform end-to-end timing analysis with the precision that is done in RCM.

#### Previous Works

In our previous works [8, 32], we presented a method to automatically extract the end-to-end timing models only at the lowest abstraction level shown in Figure 10.1. In this paper, we extend our previous method to raise the extraction of end-to-end timing models at the design level. Since, we aim to extract the timing information from a higher abstraction level, we need to explicitly capture timing requirements and constraints. For this purpose we provide unambiguous interpretation and transformation of TADL2 timing constraints in RCM.

## 10.3 Interpretation of TADL2 Timing Constraints in RCM

In the first subsection, we define some terms and notations. In the following subsections, we discuss various timing constraints in TADL2. We also discuss the semantics of each timing constraint according to the specification of TADL2 [15]. Moreover, we interpret and transform these timing constraints in RCM.

### 10.3.1 Definitions and Notations

In TADL2, timing requirements are specified by means of timing constraints on events and event chains [18]. Constraints are used to put restrictions on, e.g., delays between a pair of events; repetition of an event; and synchronicity of a set of events. An event denotes a distinct form of state change in a running system. It takes place at distinct points in time which are called its

occurrences. An event is used to trigger an analysis- or design-level function. When the function is triggered, input data is consumed followed by processing and transformation of the data and then production of the data at the output. A function can also be time-triggered. In order to clarify the notations used to define timing constraints in the following subsections, consider the following example.

#### Example

Assume a timing constraint is denoted by TC. It can be specified on some events. Let us denote two such events by source and target. We use object oriented notation to define the attributes of the constraint. For example, TC.source refers to the source event on which TC is specified. There can be any number of occurrences of an event. Let us denote an occurrence of event TC.source by an attribute s. This attribute is basically a time point when an instance of the event occurs. These time points can be added, subtracted and compared. A constraint often puts limits on the occurrences of events. These limits can be specified in terms of time distances using upper and lower attributes. In that case, the occurrences of the events are required to lie within these limits. The following provides an example for the semantics of constraint TC.

A system behavior satisfies a specified timing constraint denoted by TC iff[7] for every occurrence s of TC.source, there is an occurrence t of TC.target such that

$$\text{TC.lower} \leq (\text{t- s}) \leq \text{TC.upper}$$

This means, the timing constraint TC is satisfied if the time distance between time points t and s is greater than or equal to the time distance specified by the lower attribute; and is smaller than or equal to the time distance specified by the higher attribute.

### 10.3.2 Delay Constraint

#### TADL2 Description

It constrains the distance between occurrences of source and target events. It does not matter if matching target occurrence is caused by the corresponding source occurrence or not.

[7] if and only if

### Semantics

A system behavior satisfies the specified DelayConstraint DC iff for every occurrence s of DC.source, there is an occurrence t of DC.target such that

$$DC.lower \leq (t\text{- } s) \leq DC.upper$$

### Interpretation in RCM

There is no existing support in RCM to specify this constraint.

We propose the addition of a new timing constraint with the above mentioned semantics, denoted by Delay, in RCM. Since this constraint corresponds to the distance between occurrences of source and target events, we associate two objects with it, namely "Delay Start" and "Delay End" as shown in Figure 10.3. The Delay Start object can be specified at the Data Input Port (DIP) of source SWC. The triggering of Trigger Input Port (TIP) of source SWC corresponds to a new occurrence of source event. The triggering can be done by a clock or an event in RCM. The Delay End object can be specified at the Data Output Port (DOP) of target SWC. Production of a trigger at the Trigger Output Port (TOP) of target SWC corresponds to a new occurrence of target event. The lower and upper values of the constraint can be specified on Delay End object.

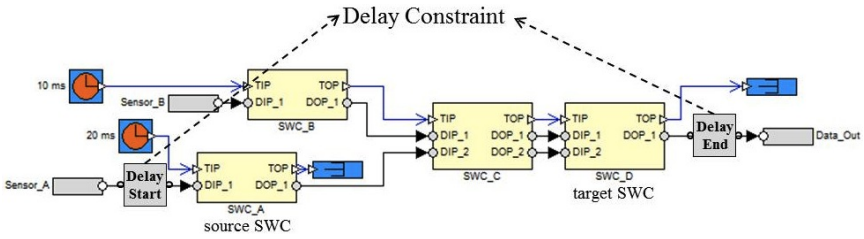


Figure 10.3: Proposed objects to specify Delay constraint in RCM

The occurrences of target event (data on DOP_1 of SWC_D) may correspond to the input data at DIP_1 of SWC_A or DIP_1 of SWC_B or both depending upon how the SWCs are triggered. In the example shown in Figure 10.3 and Figure 10.4, the occurrences of target event corresponds to input data either from SWC_B or from both SWC_A and SWC_B. The upward arrows in Figure 10.4 symbolize occurrence of events. The lower and upper attributes

for the Delay constraint are also identified in Figure 10.4. Assuming the priority of the task corresponding to SWC_A to be higher than that of SWC_B, the first occurrence of target matches the first occurrences of both SWC_B and the source. Whereas, the second occurrence of target is due to only SWC_B. As discussed earlier, matching occurrence of target with respect to occurrences of the source does not matter in this constraint.

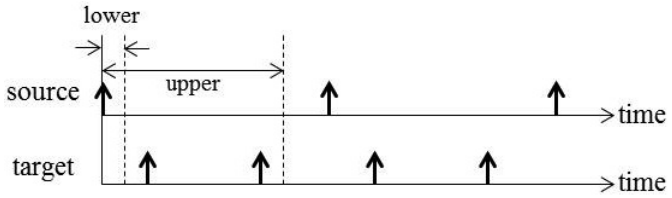


Figure 10.4: Graphical illustration of Delay constraint

### 10.3.3 Strong Delay Constraint

#### TADL2 Description

It constrains the distance between each indexed occurrence of source event and corresponding identically indexed occurrence of target event. Matching of target occurrence caused by the corresponding source occurrence is vital for this constraint.

#### Semantics

A system behavior satisfies the specified StrongDelayConstraint SDC iff the number of occurrences of SDC.source and SDC.target events is equal; and for each index i, if there is an $i^{th}$ occurrence of SDC.source at time s these is also an $i^{th}$ occurrence of SDC.target at time t such that

$$\text{SDC.lower} \leq (t - s) \leq \text{SDC.upper}$$

#### Interpretation in RCM

There is no existing support in RCM to specify this constraint.

We propose the addition of a new timing constraint with the above mentioned semantics, denoted by S-Delay, in RCM. Since this constraint corresponds to the distance between two matching occurrences of source and target events, we associate two objects with it, namely "S-Delay Start" and "S-Delay End" as shown in Figure 10.5. As the number of occurrences of source and target events for each index are not equal in the example in Figure 10.3, S-Delay constraint cannot be used in place of the Delay constraint. However, it can be used on the same system if source SWC is changed as shown in Figure 10.5. The S-Delay Start object can be specified at the DIP of source SWC. The triggering of TIP of source SWC corresponds to a new occurrence of source event. The S-Delay End object can be specified at DOP of target SWC. The production of a trigger at the TOP of target SWC corresponds to a new occurrence of target event. The lower and upper values of the constraint can be specified on S-Delay End object. These values are identified in Figure 10.6. The figure also shows that occurrences of the target match with the occurrences of the source.

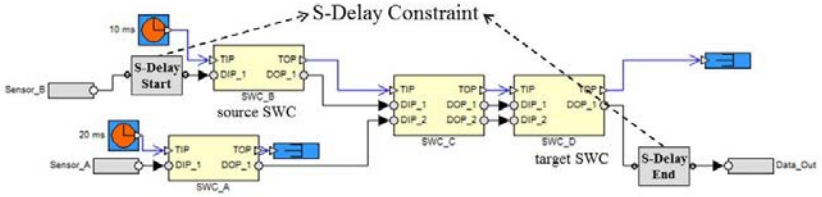


Figure 10.5: Proposed objects to specify Strong Delay constraint in RCM

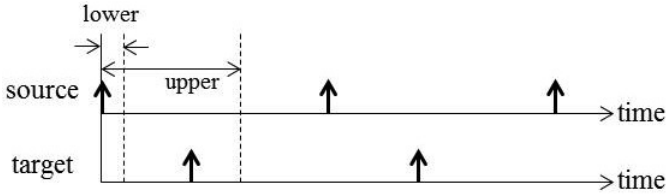


Figure 10.6: Graphical illustration of Strong Delay constraint

### 10.3.4 Reaction Constraint

#### TADL2 Description

It constrains the occurrence of a response event after the occurrence of a corresponding stimulus event in an event chain. Basically it specifies "how long after the occurrence of a stimulus a corresponding response must occur" [15]. This constraint differs from the Delay constraint in a way that it can only be applied to event chains and not to individual events.

In multi-rate systems, components in an event chain can be triggered with independent clocks. Hence, there can be multiple response occurrences to a single occurrence of stimulus in an event chain. In these chains, multiple response occurrences due to each consecutive stimulus occurrence are differentiated by means of colors. In order to satisfy this constraint, the earliest occurrence of the response with same color as that of stimulus must take place within the limits specified by this constraint as shown in Figure 10.7.

#### Semantics

A system behavior satisfies the specified ReactionConstraint ReaC iff for each occurrence s in ReaC.stimulus, there is an occurrence r in ReaC.response such that

$$\begin{gathered}(\text{r.color} = \text{s.color}) \\ \text{and} \\ (\text{r is minimal in ReaC.response with that color}) \\ \text{and} \\ (\text{minimum} \leq (\text{r- s}) \leq \text{maximum})\end{gathered}$$

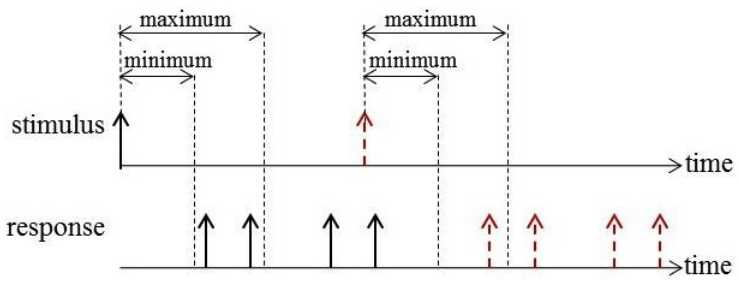


Figure 10.7: Graphical illustration of Reaction constraint

### Interpretation in RCM

There is an existing support in RCM to specify the reaction constraint denoted by DataReaction (DR for short). This constraint can be specified on an event chain, event chain segment and distributed event chain (distributed over more than one node) by means of DR Start and DR End objects as shown in Figure 10.8. The DR End object supports the specification of maximum value by means of a deadline parameter associated to it. However, the minimum parameter is considered to be zero. In order to be consistent with TADL2 Reaction constraint, we propose to associate a parameter with DR End object to specify non-zero minimum value of the constraint.

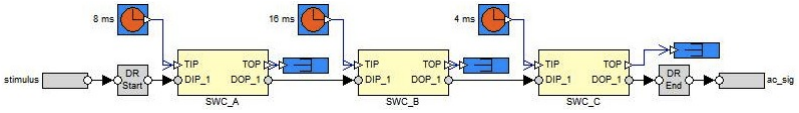


Figure 10.8: Existing objects in RCM to specify Reaction constraint

The analysis engines provided by Rubus-ICE support the calculations for the corresponding Reaction delay. Consider the example of an event chain in a multi-rate system in Figure 10.8. Figure 10.9 shows the time line when this chain is executed (assuming each SWC corresponds to a task denoted by $\tau$ at run-time). It should be noted that task_B is deliberately given an offset of 15 time units to maximize the delays. This delay is equal to the time elapsed between the previous non-overwritten release of task $\tau_A$ (input of the chain) and first response of task $\tau_C$ (output of the chain) corresponding to the current non-overwritten release of task $\tau_A$. Assume that a new value of the input is available in the input buffer of task $\tau_A$ "just after" the release of the second instance of task $\tau_A$ (at time $8ms$). Hence, the second instance of task $\tau_A$ "just misses" the read of the new value from its input buffer. This new value has to wait for the next instance of task $\tau_A$ to travel towards the output of the chain. Therefore, the new value is read by the third and forth instances of task $\tau_A$. The first output corresponding to the new value (arriving just after $8ms$) appears at the output of the chain at $34ms$. This results in the delay of $26ms$ as shown in Figure 10.9. This phenomenon is more obvious in the case of distributed embedded systems where a task in the receiving node may just miss to read fresh signals from a message that is received from the network. The analysis engines calculate the Reaction delay as shown in Figure 10.9 and compare it with the specified constraint parameters.

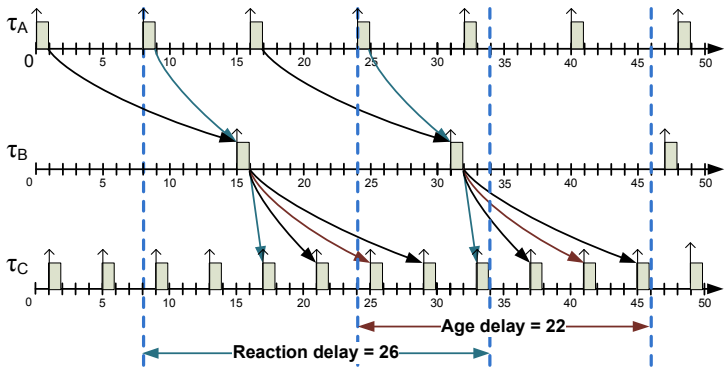


Figure 10.9: Demonstration of Reaction and Age delay calculations by analysis engines

### 10.3.5 Age Constraint

#### TADL2 Description

It constrains the occurrence of a stimulus from the occurrence of the corresponding response looking back through the event chain. Basically it specifies "how long before each response a corresponding stimulus must have occurred" [15]. In order to satisfy this constraint, the latest occurrence of the stimulus with same color as that of the response must lie within the limits specified by this constraint as shown in Figure 10.10. This constraint differs from the Delay constraint in a way that it can only be applied to event chains and not to individual events.

#### Semantics

A system behavior satisfies the specified AgeConstraint AgeC iff for each occurrence r in AgeC.response, there is an occurrence s in AgeC.stimulus such that

(s.color = r.color)
and
(s is maximal in AgeC.stimulus with that color)
and
(minimum $\leq$ (r- s) $\leq$ maximum)

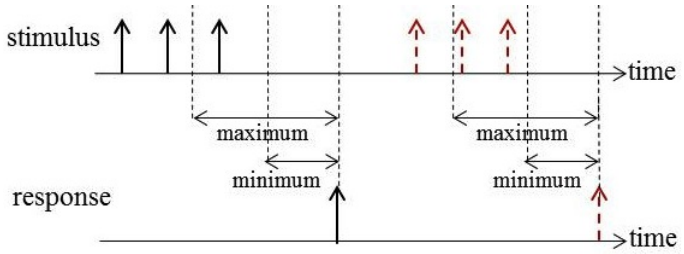


Figure 10.10: Graphical illustration of Age constraint

### Interpretation in RCM

There is an existing support in RCM to specify the age constraint denoted by DataAge. This constraint can be specified on an event chain, event chain segment and distributed event chain by means of Age Start and Age End objects as shown in Figure 10.11. The Age End object supports the specification of maximum value by means of a deadline parameter associated to it. However, the minimum parameter is considered to be zero. In order to be consistent with TADL2 Age constraint, we propose to associate a parameter with Age End object to specify non-zero minimum value of the constraint.

The analysis engines support the calculations for the corresponding Age delay. Consider the example of an event chain in a multi-rate system shown in Figure 10.11. Figure 10.9 shows the time line when this chain is executed. The analysis engines calculate the Age delay as shown in Figure 10.9 and compare it with the specified constraint parameters.

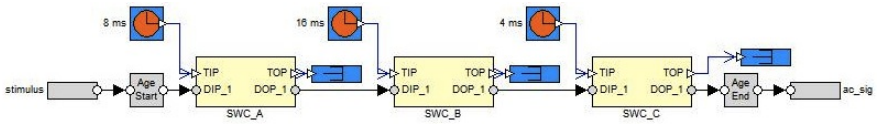


Figure 10.11: Existing objects in RCM to specify Age constraint

### 10.3.6 Repetition Constraint

#### TADL2 Description

It constrains the distribution of occurrences of a single event that may also experience *jitter* before its activation. Jitter represents the maximum variation in time with which the event can be delayed. The span attribute associated with this constraint determines which repeated occurrence will be constrained.

#### Semantics

A system behavior satisfies the specified RepetitionConstraint RC iff the following two are concurrently satisfied:

1. For each subsequence X of RC.event, if X contains span + 1 occurrences then d is the distance between the outer- and inner-most occurrences in X and

$$\text{RC.lower} \leq \text{d} \leq \text{RC.upper}$$

2. The number of occurrences of X and RC.event events is equal; and for each index i, if there is an $\text{i}^{th}$ occurrence of X at time s these is also an $\text{i}^{th}$ occurrence of RC.event at time t such that

$$0 \leq (\text{t} - \text{s}) \leq \text{RC.jitter}$$

If the span attribute is equal to one; jitter is equal to zero; and the upper and lower values are equal then the behavior becomes strictly periodic. Figure 10.12 graphically illustrates this constraint.

#### Interpretation in RCM

In RCM, an SWC can be time triggered or event triggered by means of TrigClockTT and TrigClockET objects. The TrigClockTT object generates periodic trigger signals with a period specified on it. Whereas, the TrigClockET object generates sporadic trigger signals with a minimum inter-arrival time specified on it. These two objects are shown in Figure 10.13. There is another object in RCM called TrigJitterPeriod that provides the allowance for jitter to

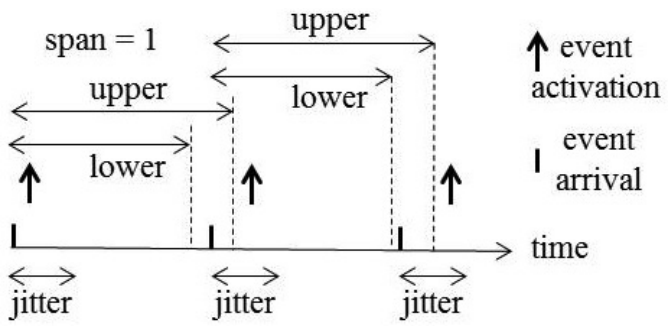


Figure 10.12: Graphical illustration of Repetition constraint

the trigger generating objects. Figure 10.13 contains two of these objects with jitter values equal to 1 millisecond and 100 microseconds.

The TrigClockTT or TrigClockET objects can be combined with TrigJitterPeriod to represent TADL2 Repetition constraint. In order to be consistent with TADL2 Repetition constraint, we propose to add span parameter to TrigClockTT and TrigClockET objects. When TrigClockTT is combined with TrigJitterPeriod, it represents TADL2 Repetition constraint with upper attribute equal to lower. When TrigClockET is combined with TrigJitterPeriod, it represents TADL2 Repetition constraint with lower and upper values assigned to minimum and maximum inter-arrival time attributes respectively.

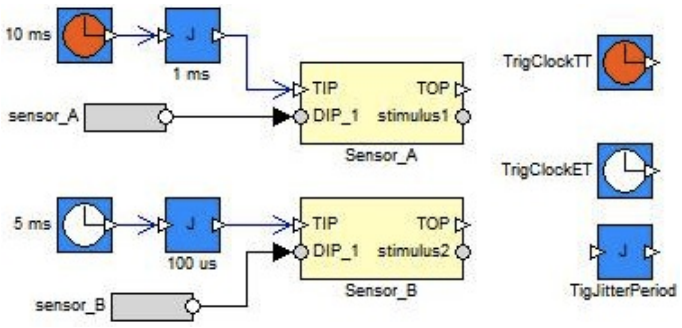


Figure 10.13: Existing objects in RCM to specify triggers and jitter

### 10.3.7 Sporadic Constraint

#### TADL2 Description

It constrains the occurrence of a sporadic event.

#### Semantics

It is a special type of Repetition constraint whose Span is equal to 1; and two subsequent activations must be separated by Minimum Inter-arrival Time (MIT). This constraint is graphically illustrated in Figure 10.14.

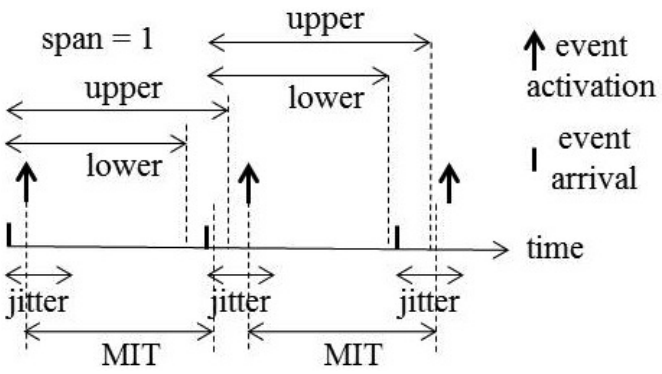


Figure 10.14: Graphical illustration of Sporadic constraint

#### Interpretation in RCM

The TrigClockET object can be combined with TrigJitterPeriod to represent TADL2 Sporadic constraint as shown in Figure 10.15. In order to consistently interpret this constraint, we set the span parameter to 1 and assign MIT value to the period associated with the TrigClockET object. The lower and upper values can be assigned to minimum and maximum inter-arrival times. If the maximum inter-arrival time is not specified, it can be considered equal to infinity.

### 10.3.8 Periodic Constraint

#### TADL2 Description

It constrains the occurrence of a periodic event.

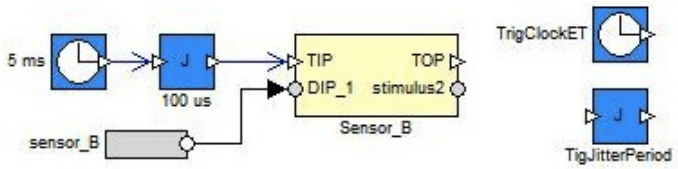


Figure 10.15: Equivalent to Sporadic constraint specified in RCM

### Semantics

It is a special type of Sporadic constraint whose lower and upper attributed are equal. These attributes are assigned the value of the period. This constraint is graphically illustrated in Figure 10.16.

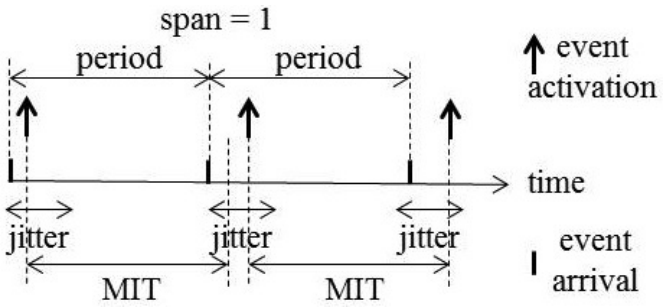


Figure 10.16: Graphical illustration of Periodic constraint

### Interpretation in RCM

The TrigClockTT object can be combined with TrigJitterPeriod to represent TADL2 SporadicConstraint as shown in Figure 10.17. In order to consistently interpret this constraint, we set the span parameter to 1; upper and lower parameters are equal and are assigned the value of period; and assign MIT value to the period associated with the TrigClockTT object unless specified.

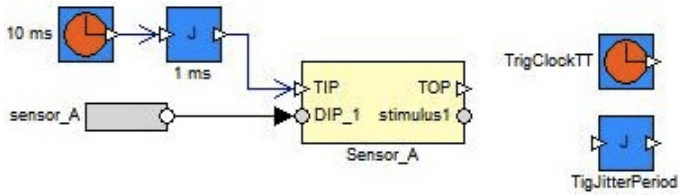


Figure 10.17: Equivalent to Periodic constraint specified in RCM

### 10.3.9 Pattern Constraint

#### TADL2 Description

It constrains the occurrences of an event that follow a certain pattern with respect to some periodic temporal points.

#### Semantics

A system behavior satisfies the specified PatternConstraint PC iff there is a set of times X such that the same system behavior concurrently satisfies the following conditions:

1. PeriodicConstraint with a period equal to PC.period
2. For each PC.offset index i: for every occurrence x of X, there is an occurrence t of the PC.event such that

$$\text{PC.offset}_i \leq \text{(t- x)} \leq \text{(PC.offset}_i + \text{PC.jitter)}$$

3. If X contains two occurrences then d is the distance between the outer- and inner-most occurrences in X and

$$\text{PC.minimum} \leq \text{d}$$

This constraint is graphically illustrated in Figure 10.18. In each period of event patterns, the event occurrences happen at predefined temporal points called offsets with respect to the starting reference point in that period. Each occurrence of the event can be influenced by the specified jitter.

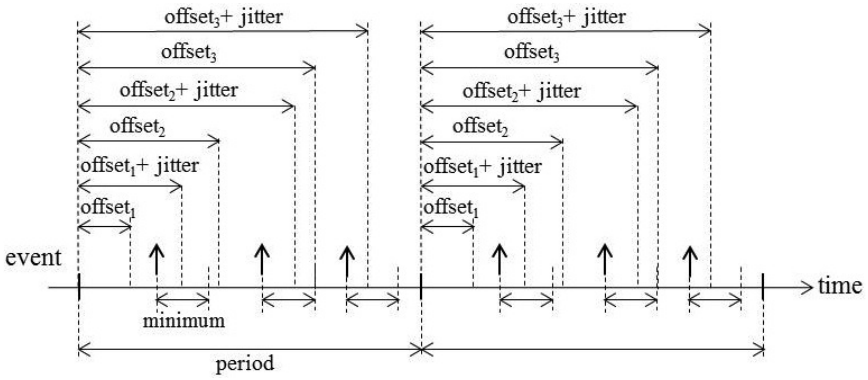


Figure 10.18: Graphical illustration of Pattern constraint

### Interpretation in RCM

This constraint is similar to the transactional model of tasks with offsets which is inherent to the time-triggered execution in RCM. At run-time, all time triggered tasks (assuming an SWC corresponds to a task at run-time) from a node are combined into one big transaction with a period while the tasks have offsets and jitter. The period of the transaction is the least common multiple of the periods of all tasks in the transaction.

We propose the addition of a new timing constraint with the above mentioned semantics, denoted by Pattern constraint, in RCM as shown in Figure 10.19. The parameters associated to this object are period, minimum inter-arrival time, jitter, number of event occurrences during the period time and a set of offsets. The analysis engines are responsible to satisfy this constraint by comparing the specified parameters with the corresponding parameters in the transactional model.

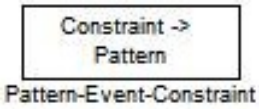


Figure 10.19: Proposed object in RCM to specify Pattern constraint

### 10.3.10 Arbitrary Constraint

#### TADL2 Description

It constrains an event that occurs irregularly. It contains a set of pairs consisting of minimum inter-arrival time (denoted by min) and maximum inter-arrival time (denoted by max).

#### Semantics

A system behavior satisfies the specified ArbitraryConstraint AC iff for each AC.min index i, the same system behavior satisfies, for each subsequence X of AC.event, if X contains i + 1 occurrences then d is the distance between the outer- and inner-most occurrences in X and

$$\text{AC.min}_i \leq \text{d} \leq \text{AC.max}_i$$

The constraint is graphically illustrated in Figure 10.20. In the figure, min1, min2 and min3 represent minimum inter-arrival time between/among two, three and four subsequent occurrences of the event respectively. Similarly, max1, max2 and max3 represent maximum inter-arrival time between/among two, three and four subsequent occurrences of the event respectively. Although, three pairs of min and max parameters are plotted for first two occurrences of the event, these parameters continue in a similar fashion for the rest of occurrences of the event.

#### Interpretation in RCM

There is no existing support to specify the arbitrary constraint in RCM.

We propose the addition of a new timing constraint with the above mentioned semantics, denoted by Arbitrary constraint, in RCM as shown in Figure 10.21. It is able to specify any number of pairs of min and max values.

### 10.3.11 Execution Time Constraint

#### TADL2 Description

It constrains the time between activation and completion of the execution of a function (executable entity). However, the intervals, when the execution of the function is interrupted due to preemptions and blocking, are not considered in this constraint.

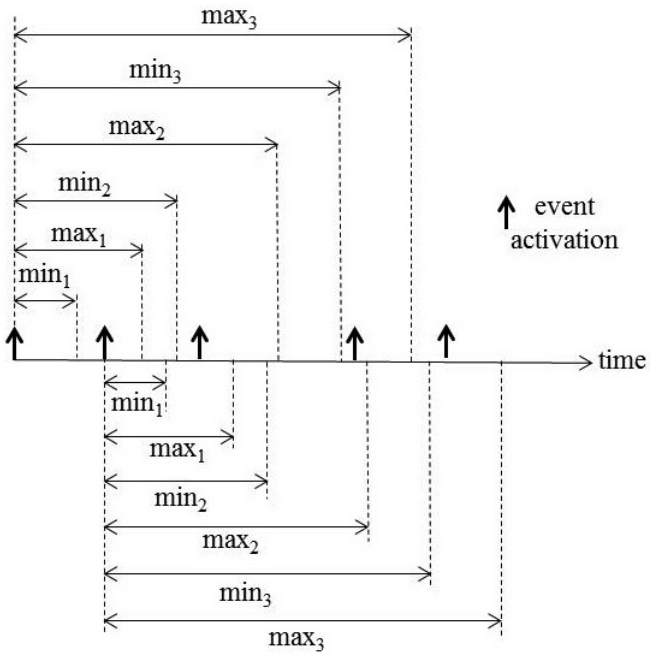


Figure 10.20: Graphical illustration of Arbitrary constraint


Figure 10.21: Proposed object in RCM to specify Arbitrary constraint

**Semantics**

A system behavior satisfies the specified ExecutionTimeConstraint ETC iff for each occurrence x of event ETC.activate, $ET_i$ is the set of times between x and the next ETC.completion while excluding the times due to ETC.preemtion and ETC.blocking, and that

$$\text{ETC.lower} \leq \text{sum of all continuous intervals in } ET_i \leq \text{ETC.upper}$$

This constraint is graphically illustrated in Figure 10.22.

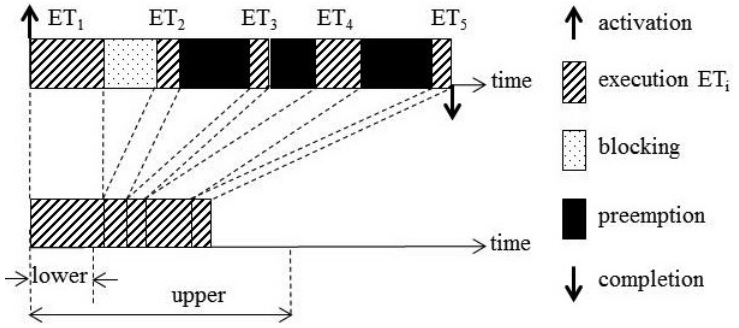


Figure 10.22: Graphical illustration of Execution Time constraint

#### Interpretation in RCM

There is an equivalent existing support in RCM to specify the execution time constraint for an SWC. Each SWC has one or more behaviors, whereas, each behavior represents a function. When an SWC is triggered, its state and data (from all of its DIPs) are passed to it. The states are updated and newly calculated data is placed on the DOPs while a trigger is produced at the TOP upon completion of the behavior. RCM supports the specification of three types of execution times on the behavior of SWC namely Best Case Execution Time (BCET), Worst Case Execution Time (WCET) and Average Case Execution Time (ACET) as shown in Figure 10.23. In order to unambiguously interpret this constraint in RCM, the lower and upper values of this constraint (see Figure 10.22) can be assigned to the BCET and WCET parameters respectively in Figure 10.23.

### 10.3.12 Synchronization Constraint

#### TADL2 Description

It constrains the closeness of the occurrences of a group of events.

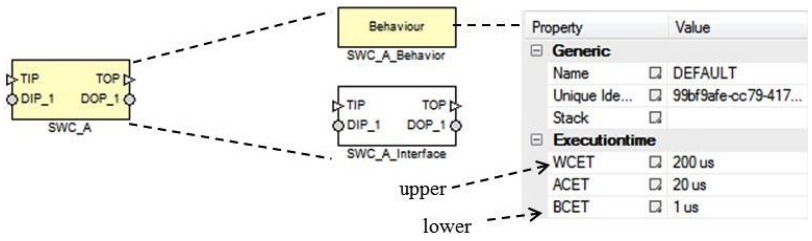


Figure 10.23: Equivalent to Execution Time constraint specified in RCM

### Semantics

A system behavior satisfies the specified SynchronizationConstraint on a given set of events and given the occurrence of any event in this set, then the rest of the events in the set must occur at least once within a certain time window called tolerance.

This constraint is graphically illustrated in Figure 10.24. It is applied on the two events data_A and data_B. In this constraint, more than one instance of the events may exist in a time window provided the above conditions are met. Moreover, the windows may overlap and share occurrences of the events.

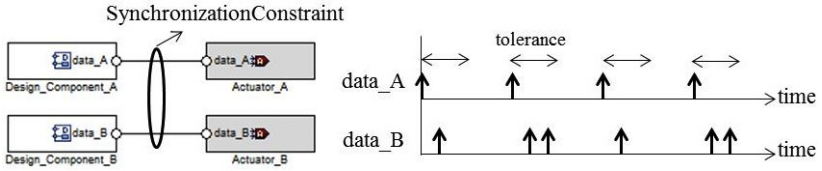


Figure 10.24: Graphical illustration of Synchronization constraint

### Interpretation in RCM

There is an existing support in RCM to synchronize multiple triggers by means of a synchronization object denoted by TrigSync as shown in Figure 10.25. This object has two or more TIPs and only one TOP. The synchronization condition can use either AND or OR semantics. In the case of AND condition, the TOP is triggered only when trigger signals have arrived at all TIPs. Whereas, in the case of OR condition, the TOP is triggered as soon as there is a trig-

ger signal at one of the TIPs. In order to make it consistent with the TADL2 Synchronization constraint, we add the tolerance parameter to this object. The analysis engine must ensure that this constraint is satisfied within the tolerance window.

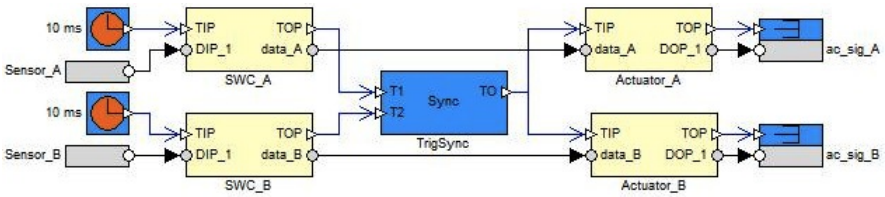


Figure 10.25: Synchronization constraint in RCM

## 10.3.13 Strong Synchronization Constraint

### TADL2 Description

It constrains the closeness of the occurrences of a group of events.

### Semantics

The semantics of this constraint differ from the SynchronizationConstraint in a way that the occurrences of the events in a window must have same indices. Therefore, more than one instance of the events cannot exist in a time window. Moreover, the windows cannot overlap and share occurrences of the events.

This constraint is graphically illustrated in Figure 10.26. It is applied on the two events data_A and data_B.

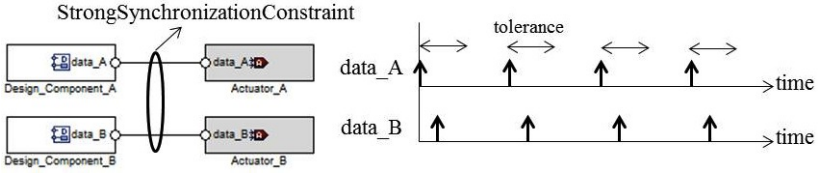


Figure 10.26: Graphical illustration of Strong Synchronization constraint

### Interpretation in RCM

There is an existing support in RCM to synchronize multiple triggers by means of a synchronization object denoted by TrigSync. In order to differentiate the strong synchronization constraint from this object, we propose to add a similar object denoted by S-TrigSync as shown in Figure 10.27. This object has two or more TIPs and only one TOP. The synchronization condition can use either AND or OR semantics. In order to make it consistent with the TADL2 Strong Synchronization constraint, we add the tolerance parameter to this object.

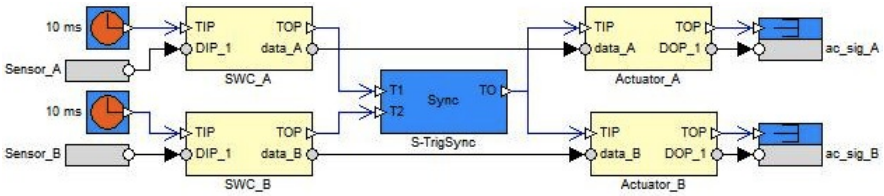


Figure 10.27: Proposed object in RCM to specify Strong Synchronization constraint

## 10.3.14 Output Synchronization Constraint

### TADL2 Description

It constrains the closeness of the occurrences of responses to a certain stimulus. Basically, it defines how far apart the responses to a certain stimulus can occur. This constraint differs from the SynchronizationConstraint in a way that it can only be applied to a set of event chains such that there are multiple responses to a single stimulus as shown in Figure 10.28 and Figure 10.29. The tolerance parameter constrains the latest of these response occurrences for each chain. The system in Figure 10.28 is modeled with two event chains. They have common stimulus but different responses denoted by response1 and response2.

### Semantics

A system behavior satisfies the specified OutputSynchronizationConstraint OSC iff for each occurrence s in OSC.stimulus, there is a time t such that for each index i, there is an occurrence r in OSC.response$_i$ such that

$$(r.color = s.color)$$

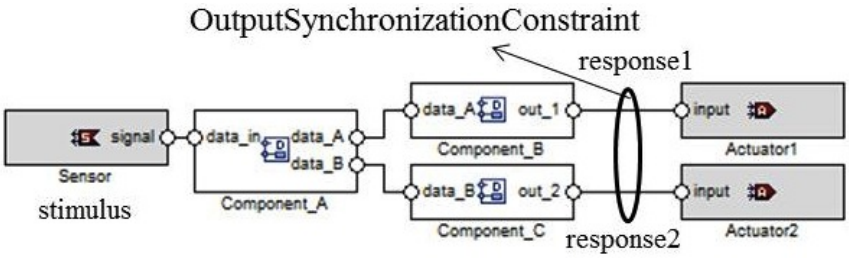


Figure 10.28: Usage of Output synchronization constraint at the design level

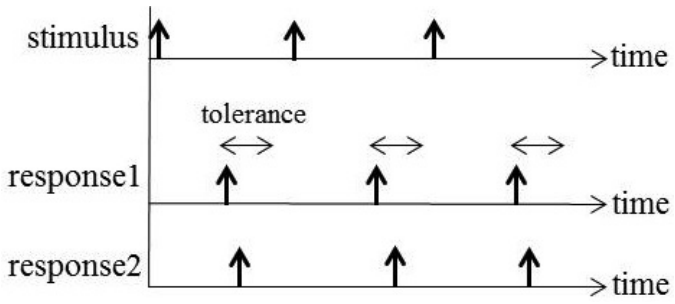


Figure 10.29: Graphical illustration of Output Synchronization constraint

and
(r is minimal in OSC.response$_i$ with that color)
and
($0 \leq$ (r- t) $\leq$ OSC.tolerance)

**Interpretation in RCM**

There is an existing support in RCM to synchronize multiple triggers by using TrigSync object. We propose to add a similar object, denoted by Out-TrigSync, in RCM. This object has two or more TIPs and only one TOP. The synchronization condition can use either AND or OR semantics. In order to make it consistent with the TADL2 Output Synchronization constraint, we add the tolerance parameter to it. The analysis engine must ensure that this constraint is satisfied within the tolerance window. The example in Figure 10.30 depicts a single rate system; hence there cannot be more than one occurrences of each

response corresponding to single occurrence of the stimulus. However, Out-TrigSync is equally applicable to multi-rate systems where the components are triggered with independent clocks.

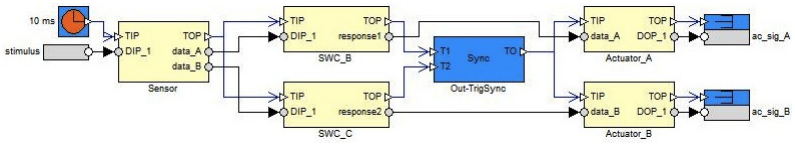


Figure 10.30: Proposed object to specify Output Synchronization constraint in RCM

### 10.3.15 Input Synchronization Constraint

#### TADL2 Description

It constrains the closeness of the occurrences of stimuli corresponding to a certain response. Basically, it defines how far apart the stimuli corresponding to a certain response can occur. This constraint differs from the Synchronization constraint in a way that it can only be applied to a set of event chains such that there are multiple stimuli and a single corresponding response as shown in Figure 10.31 and Figure 10.32. The tolerance parameter constrains the latest of these stimuli occurrences for each chain. This means that once one of the stimuli has been acquired, the others should be acquired within a time window equal to the tolerance parameter. The system in Figure 10.31 is modeled with two event chains. They are initiated by separate stimuli but have one common response.

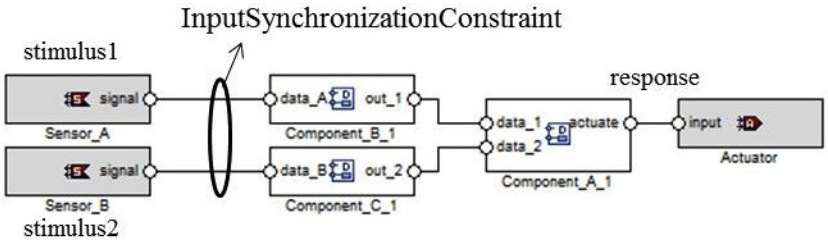


Figure 10.31: Usage of Input synchronization constraint at the design level

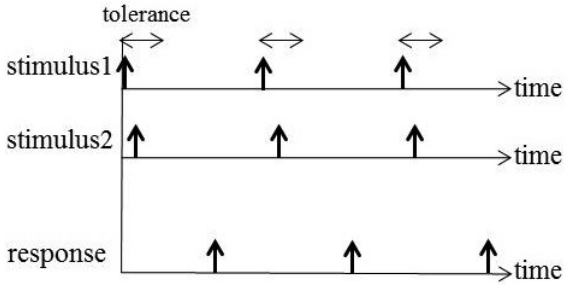


Figure 10.32: Graphical illustration of Input Synchronization constraint

### Semantics

A system behavior satisfies the specified InputSynchronizationConstraint ISC iff for each occurrence r in ISC.response, there is a time t such that for each index i, there is an occurrence s in ISC.stimulus$_i$ such that

$$\begin{gathered}\text{(r.color = s.color)}\\ \text{and}\\ \text{(s is minimal in ISC.stimulus}_i\text{ with that color)}\\ \text{and}\\ \text{(0} \leq \text{(s- t)} \leq \text{ISC.tolerance)}\end{gathered}$$

### Interpretation in RCM

There is an existing support in RCM to synchronize multiple triggers by using TrigSync object. We propose to add a similar object, denoted by In-TrigSync, in RCM. This object has two or more TIPs and only one TOP. The synchronization condition can use either AND or OR semantics. In order to make it consistent with the TADL2 Input Synchronization constraint, we add the tolerance parameter to it. The example in Figure 10.33 depicts a single rate system; hence there cannot be more than one occurrences of each response corresponding to single occurrence of the stimulus. However, In-TrigSync is equally applicable to multi-rate systems where the components are triggered with independent clocks.

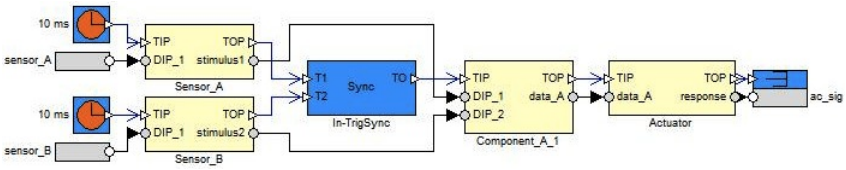


Figure 10.33: Proposed object to specify Input Synchronization constraint in RCM

## 10.4 Discussion

The models and approaches that are used at the implementation level such as RCM and AUTOSAR support the extraction of end-to-end timing models. However, the modeling approaches used at the design or higher levels such as EAST-ADL, TIMMO and TADL2 do not support complete and unambiguous extraction of the timing models. Due to unavailability of the end-to-end timing model at higher abstraction levels, it may be impossible to perform the timing analysis in some cases.

We focus on the design level within the context of this problem. We consider the modeling support of EAST-ADL, TIMMO and TADL2 at the design level. Whereas, the modeling support of RCM is considered at the implementation level. We discuss some of the challenges that hinder the extraction of the end-to-end timing model. We also propose guidelines and solutions to deal with these challenges. We also discuss a proposal for implementation of the solutions in RCM.

### 10.4.1 Extraction of Control and Data Paths

Unambiguous extraction of control (trigger) and data paths from the system are vital for performing its end-to-end timing analysis. A trigger path captures the flow of triggers along a chain of components (tasks at run-time). For example, trigger path of the chain shown in Figure 10.34(c) can be expressed as $\{\{SWC\_A \rightarrow SWC\_B\}, \{SWC\_C\}\}$ because SWC_B is triggered by SWC_A, while SWC_C is triggered independently. Similarly, trigger paths of the chains shown in Figure 10.34(a) and Figure 10.34(b) can be expressed as $\{\{SWC\_A \rightarrow SWC\_B \rightarrow SWC\_C\}$ and $\{\{SWC\_A\}, \{SWC\_B\}, \{SWC\_C\}\}$ respectively.

One of the main challenges in the extraction of the timing model at the design level is the lack of clear separation between the trigger and data paths. At

the implementation level, e.g. in RCM, these paths are clearly separated from each other by means of trigger and data ports as shown in Figure 10.35(b). A TOP of an SWC can only be connected to the TIP(s) of other SWC(s). Similarly, a DOP of an SWC can only be connected to the DIP(s) of other SWC(s). Hence, the trigger and data paths can be clearly identified.

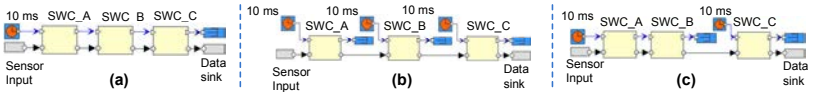


Figure 10.34: Example of (a) Trigger chain, (b) Data chain, and (c) Mixed chain.

On the other hand, at the design level, the components communicate via *flow ports* as shown in Figure 10.35(a). A flow port is an EAST-ADL object that is used to transfer data between components. It is single buffer, non-consumable and over-writable. Without any explicit information, it can be interpreted as a data or trigger port at the implementation level. There is no support to specify explicit trigger paths at the design level. Moreover, a component can be triggered via specified timing constraints on event, modes, or internal behavior of the component. The two types of flows should be clearly and separately captured in the end-to-end timing model because the type of the timing analysis depends upon it. For example, it is not meaningful to perform end-to-end delay analysis on a trigger chain shown in Figure 10.35(a) [5].

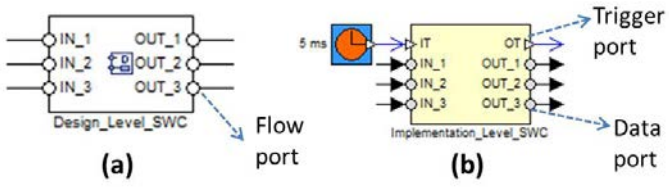


Figure 10.35: Model of the SWC at (a) design level, (b) implementation level

In order to clearly identify the trigger and data paths at the design level, we make some assumptions.

1. We assume a one-to-one mapping between each design- and implementation level component. Although, a design-level component can be

mapped to more than one implementation-level components; our assumption is based on common practice that is used in the industry, especially in the segment of construction-equipment vehicles domain.

2. If a timing constraint is specified on the flow port, we assume the component is triggered independently. The type of triggering is judged by the type of specified constraint.

3. If Age or Reaction constraint is specified on a chain; and no other constraint is specified, we assume that the first and last components in the chain are triggered independently. This is because more than one independent trigger in a chain makes it either data or mixed chain. It is meaningful to specify the Age and Reaction constraints only on data and mixed chains.

4. We assume that a flow port is implicitly triggered at the arrival of data. If there are more than one flow ports in a component, arrival of data at each port produces a trigger. For example, the component in Figure 10.35(a) may receive three individual triggers when data is separately received at three input flow ports. The TrigSync object in RCM can be used to deal with multiple implicit triggers (corresponding to multiple flow ports) at the implementation level. This object gets the multiple triggers at input, synchronizes them, and produces a single trigger that can be used to trigger SWC (corresponding to the design-level component) at the implementation level. Figure 10.36 shows implementation-level equivalent of design-level component with three flow ports as shown in Figure 10.35(a).

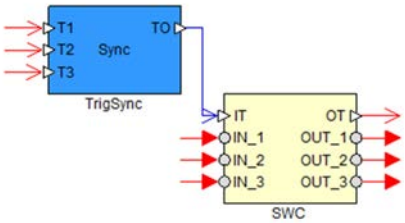


Figure 10.36: Implementation-level equivalent of design-level component in Figure 10.35(a)

### 10.4.2 Annotation and Extraction of Timing Parameters

The timing information expressed with the models and tools used at the design level is not enough to extract the end-to-end timing model. For example, one of the EAST-ADL based tools[8] used at the design and higher levels is able to specify only one timing parameter on components, i.e., the period of the component. Clearly, this information is not enough to perform the end-to-end timing analysis. TADL2 can specify timing constraints and properties at the design level in EAST-ADL and AUTOSAR based development. However, it lacks the expression of some timing parameters, e.g., priority and transmission type which are needed to perform the end-to-end timing analysis. We already discussed the interpretation of TADL2 timing constraints in RCM in the previous section.

We assume that the execution order of design-level components in a chain is specified, otherwise, we make implicit assumption about it. That is, each component is assumed to execute only after successful execution of preceding component in the chain, unless specified otherwise. This means, a data provider component is assumed to be always executed before the data receiver component. Since, this assumption fixes the execution order, it is safe to assume the priorities of the components are equal within the chain. If worst-, best- and average-case execution times are not available at the design level, they can be estimated at the implementation level either using estimations by the experts or reusing from other projects.

### 10.4.3 Identification of Chain Types

Since, control and data flows are clearly separated at the implementation levels, e.g., in RCM, the chain types can be easily identified as shown in Figure 10.34. Due to no clear separation between these flows at the design level, virtually it is not possible to identify the type of a chain. At the design level, a chain can be interpreted as a trigger or data chain. Without any explicit trigger information, the end-to-end timing analysis cannot be performed. This is because trigger chains are analyzed using end-to-end response-time analysis, whereas, data and mixed chains are analyzed using both end-to-end response-time and delay analyses [5]. If there are no constraints specified on a chain, we assume it to be a trigger chain. Otherwise, it can be considered as a data or a mixed chain depending upon how the constraints are specified.

[8] For IP protection, the name of the tool is not specified.

### 10.4.4 Information Duplication and Ambiguity

At the implementation level, for example, RCM does not allow illogical operations such as specifying more than one clock on the same component without any synchronization or merge operation. However, these restrictions are not present at the design level, e.g., more than one execution time or periodic constraint can be specified on a single component in EAST-ADL using TADL2. Similarly, if data age and reaction constraints are wrongly specified then the development environment does not complain about it. As a result, the extracted timing model may have redundant or erroneous information. Information duplication can lead to inconsistency in the timing model. However, at the implementation level, Rubus-ICE complains about these inconsistencies and ambiguities. The analysis engines calculate delay and reaction constraints only when they are specified on data and mixed chains.

### 10.4.5 Conclusion

There can be two different approaches to deal with these challenges. The first approach is to extend and improve the design-level models, languages and tools in such a way that the timing models can be completely and unambiguously extracted. Moreover, the extracted models are general enough to be operated by different models and tools. The only problem with this approach is that it requires strong collaboration among a number of tool suppliers and stake holders. This, in turn, raises other types of challenges and limitations. The second approach is to develop the execution-level modeling technology-dependent interpretation of the design level. For example, developing Rubus interpretation of EAST-ADL (this is an ongoing work). It is important to note that this interpretation can be a subset of the full expressiveness of EAST-ADL. No doubt, this may result in a number of these interpretations by several other modeling technologies. This can be a good solution as long as these interpretations support unambiguous extraction of end-to-end timing models. We propose to implement the second option.

## 10.5 Summary and Future Work

We extended our previous method to support the extraction of end-to-end timing models at a higher level of abstraction. The purpose is to support the end-to-end timing analysis at a higher abstraction level and early phases during the development of component-based vehicular distributed embedded systems.

At the higher level, the method extracts timing information from models of the systems that are developed with EAST-ADL and TADL2 languages using TIMMO methodology. Whereas, at the lower level, it considers Rubus Component Model (RCM) to extract the timing information that cannot be clearly specified at the higher level. As part of this method, we provided an interpretation of TADL2 timing constraints in RCM. We also proposed extensions in RCM for unambiguous transformation of these constraints. Moreover, we discussed the challenges and issues that are faced during the extraction of end-to-end timing information at a higher abstraction level. Further, we presented guidelines and solutions to deal with these challenges. These challenges and corresponding solutions may be equally applicable for other modeling technologies suitable for these abstraction levels.

In the future we plan to conduct an industrial case study to provide a proof of concept for the end-to-end timing model extraction method at various abstraction levels. In TADL2, time can be expressed in multiple time bases, e.g., chronometric time; angular time; revolution per minute; and time expressed in distance or rotation of crank shaft. Furthermore, time can also be expressed as algebraic expressions and parameterized expressions between different time bases using the Symbolic Timing Expression [15]. It can be an interesting future work to provide support for timing expressions based on these multiple time bases in RCM.

## Acknowledgement

The work in this paper is supported by the Swedish Research Council (VR) within the project SynthSoft. We thank our industrial partners Arcticus Systems and Volvo, Sweden.

# Bibliography


[1] Peter Thorngren. Keynote Talk: Experiences from EAST-ADL Use, EAST-ADL Open Workshop, Gothenberg, Oct., 2013.

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

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

[5] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Support for end-to-end response-time and delay analysis in the industrial tool suite: Issues, experiences and a case study. *Computer Science and Information Systems, ISSN: 1361-1384*, 10(1), 2013.

[6] AUTOSAR Techincal Overview, Release 4.1, Rev. 2, Ver. 1.1.0., The AUTOSAR Consortium, Oct., 2013. http://autosar.org.

[7] Sverine Sentilles, Aneta Vulgarakis, Tomas Bures, Jan Carlson, and Ivica Crnkovic. A Component Model for Control-Intensive Distributed Embedded Systems. In *11th International Symposium on Component Based Software Engineering*, pages 310–317. Springer, Oct. 2008.

[8] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Communications-Oriented Development of Component- Based Vehicular Distributed Real-Time Embedded Systems. *Journal of Systems Architecture*, 2013.

[9] 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, Aug. 2007.

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

[11] TIMMO Methodology, Ver. 2. *TIMMO (TIMing MOdel), Deliverable 7*, Oct. 2009. The TIMMO Consortium.

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

[13] CRYSTAL - CRitical sYSTem engineering AcceLeration, http://www.-crystal-artemis.eu, accessed May, 2014.

[14] OMG Systems Modeling Language, version 1.3. http://www.omgsysml.-org.

[15] Timing Augmented Description Language (TADL2) syntax, semantics, metamodel Ver. 2, Deliverable 11, Aug. 2012.

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

[17] EAST-ADL Domain Model Specification, Deliverable D4.1.1. http://www.atesst.org/home/liblocal/docs/ATESST2_D4.1.1_EAST-ADL2-Specification_2010-06-02.pdf.

[18] TIMMO-2-USE Methodology Description, Ver. 2, Del. 13, Jul., 2012.

[19] Rubus ICE-Integrated Development Environment. http://www.arcticus-systems.com.

[20] PeterH. Feiler, Bruce Lewis, Steve Vestal, and Ed Colbert. An Overview of the SAE Architecture Analysis & Design Language (AADL) Standard: A Basis for Model-Based Architecture-Driven Embedded Systems Engineering. In *Architecture Description Languages*, volume 176 of *The International Federation for Information Processing (IFIP)*, pages 3–15. Springer US, 2005.

[21] SCADE Suite, http://www.esterel-technologies.com/products/scade-suite, accessed May, 2014.

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

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

[24] CHESS Project, CHESS consortium. Avialable at: http://www.chess-project.org, accessed May, 2014.

[25] Antonio Cicchetti, Federico Ciccozzi, Silvia Mazzini, Stefano Puri, Marco Panunzio, Tullio Vardanega, and Alessandro Zovi. Chess: a model-driven engineering tool environment for aiding the development of complex industrial systems. In *27th International Conference on Automated Software Engineering (ASE 2012)*, Sep. 2012.

[26] TADL: Timing Augmented Description Language, Ver. 2, Deliverable 6, Oct., 2009.

[27] Hans Blom et. al. EAST-ADL- An Architecture Description Language for Automotive Software-Intensive Systems. White paper, Ver. M2.1.10, 2012, http://www.maenad.eu.

[28] Rubus models, methods and tools. http://www.arcticus-systems.com.

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

[30] Mastering Timing Information for Advanced Automotive Systems Engineering. In the TIMMO-2-USE Brochure, 2012. Available at: http://www.timmo-2-use.org/pdf/T2UBrochure.pdf.

[31] Jan Carlson. Timing Analysis of Component-based Embedded Systems. In *15th International ACM SIGSOFT Symposium on Component Based Software Engineering*. ACM, Jun. 2012.

[32] 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, Oct. 2011.