2012 IEEE 19th International Conference and Workshops on Engineering of Computer-Based Systems

Towards Modeling and Holistic Timing Analysis of Industrial Component-Based DRE Systems

Saad Mubeen*, Jukka Mäki-Turja*† and Mikael Sjödin*
* *Mälardalen Real-Time Research Centre (MRTC), Mälardalen University, Västerås, Sweden*
† *Arcticus Systems, Järfälla, Sweden*
{*saad.mubeen, jukka.maki-turja, mikael.sjodin*}*@mdh.se*

***Abstract*—We propose a model- and component-based approach for communications-oriented development of Distributed Real-time Embedded (DRE) systems with a support for legacy communication protocols, legacy nodes and Holistic Response Time Analysis (HRTA). Because an end-to-end timing model should be available to perform HRTA, we also present a method to extract such models from component-based DRE systems. Moreover, we extend the existing analysis of Controller Area Network to support mixed messages in the system with priority- and FIFO-queued nodes. A mixed message represents a common transmission pattern implemented by some high-level protocols used in the industry. We also provide a proof of concept by extending the existing industrial model, i.e., Rubus Component Model, implementing the HRTA along with the extended analysis in an industrial tool suite, i.e., Rubus-ICE, and conducting an automotive-application case study.**

***Keywords*-Distributed real time embedded systems; DRE systems; holistic response-time analysis; component-based development; component model; timing model; real-time analysis.**

I. Introduction

In this paper, we present the development and implementation of new modeling and timing analysis techniques to support the state-of-the-practice development of component-based Distributed Real-time Embedded (DRE) systems.

A. Background

Embedded systems are found in almost all electronic products around us. Their applications span over many domains including automotive, aerospace, consumer electronics, biomedical, military, business, industrial control, etc. 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 (embedded software) has drastically increased in size and complexity. In automotive domain, e.g., a modern premium car contains nearly 100 million lines of code that run on about 70 to 100 embedded processors [3].

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

In order to capture, e.g., requirements early during the development, handle the complexity of embedded software, lower the development cost, reduce the time-to-market and time-to-test, allow reusability, and support modeling at higher level of abstraction, the research community proposed model- and component-based development of embedded systems by employing the principles of Model-Based software Engineering (MBE) and Component-Based Software Engineering (CBSE) [4], [5]. MBE provides the means to use models throughout the process of system development. It uses models to describe functions, structures and other design artifacts. Whereas, CBSE 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. There is a great interest for bringing these techniques in the embedded systems industry [5], [6].

In DRE systems, the functionality is distributed over many nodes (processors) that communicate with each other via one or more networks. The software development of DRE systems is much more complex compared to uniprocessor embedded real-time systems because of various reasons including the distribution of functionality and real-time requirements on network communications. When MBE and CBSE are used for the development of DRE systems, modeling of communication infrastructure arises as a challenge. In the industry, DRE systems are built often using legacy (sub) systems (i.e., previously developed) which use predefined

978-0-7695-4664-3/12 $26.00 © 2012 IEEE
DOI 10.1109/ECBS.2012.46

rules for communication. DRE systems are often expected to use legacy protocols for network communication. A component technology for the development of DRE systems should abstract the application software from the communication infrastructure and support the modeling and analysis of legacy communications and legacy systems.

The need for safety criticality in many DRE systems requires evidence that each action by the system will be taken at a time that is appropriate to its environment. It is important to make accurate predictions of the timing behavior of such systems. In order to provide evidence that each action in the system will meet its deadline, *a priori* analysis techniques such as schedulability analysis have been developed by the research community. The Holistic Response-Time Analysis (HRTA) [7] is a schedulability analysis technique to calculate upper bounds on the response times of event chains that are distributed over more than one node in a DRE system. The end-to-end timing model of a DRE system should be available to perform HRTA. Ideally, a component technology for the development of DRE systems should support automatic extraction of such models.

There are a number of real-time network protocols used in DRE systems. Among them, Controller Area Network (CAN) [8] is one of the most frequently used especially in automotive domain. It has been standardized by the International Organization for Standardization as ISO 11898-1 [9]. According to CAN in Automation (CiA) [10], the number of CAN enabled controllers sold in 2011 are estimated to be 850 million. In total, more than two billion CAN controllers have been sold until today. Out of this huge number, approximately 80% CAN controllers have been used in automotive domain. CAN is a multi-master, event-triggered, serial communication bus protocol supporting bus speeds of up to 1 mega bits per second. In this paper, we will focus only on CAN and some of its high-level protocols which are developed for various industrial applications. These include CAN Application Layer (CAL) [11], CANopen [12], Hägglunds Controller Area Network (HCAN) [13], CAN for Military Land Systems domain (MilCAN) [14], etc.

B. Paper Layout

The rest of the paper is organized as follows. In Section II, we formulate the research problem. Section III describes the Rubus concept. In Section IV, we present the related work. Section V discusses the paper contributions. Finally, Section VI concludes the paper.

II. Problem Formulation

The model- and component-based development has emerged as an attractive option for the development of software for DRE systems. The majority of existing model- and component-based development approaches allow for structural and functional modeling. They do not support execution modeling which is concerned with the modeling of run-time properties and/or requirements (e.g., end-to-end deadlines, jitter, etc.) of software functions. The modeling of DRE systems should extend down to the execution level to allow precise control of resource utilization and that timing requirements are not violated when the system is executed. However, providing such modeling support for DRE systems is very challenging because the functionality in DRE systems can be realized with more than one execution model, e.g., separate execution models for the nodes and networks. Today, one of the main focus points during the development of DRE systems in the industry is to model and express timing related information and perform timing analysis [15].

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

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

In order to provide a model- and component-based approach to support the state-of-the-practice development of DRE systems, we will target the challenges concerned with the modeling of real-time network communication and

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

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

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

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

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

III. 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 [16] in close collaboration with several academic and industrial partners. Rubus is today mainly used for development of control functionality in vehicles. The Rubus concept is based around the Rubus Component Model (RCM) [17] 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.

A. The Rubus Component Model (RCM)

RCM expresses the infrastructure for software functions, i.e., the interaction between the software functions in terms of data and control flow separately. The control flow is expressed by triggering objects 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, i.e., explicit synchronization or data access should all be visible at the modeling level.

In RCM, the basic component is called a Software Circuit (SWC). It is the lowest-level hierarchical element in RCM and its purpose is to encapsulate basic functions. The SWCs interact with each other through the use of ports. An SWC can be seen as a type, or a class, that can be instantiated an arbitrary number of times. By separating functional code and the infrastructure, RCM facilitates analysis and reuse of components in different contexts (an SWC has no knowledge how it connects to other components). The execution semantics of software components (functions) is simply: upon triggering, read data on data in-ports; execute the function; write data on data out-ports; and activate the output trigger. Furthermore, the component model has a possibility to encapsulate SWCs into software assemblies enabling the designer to construct the system at different hierarchical levels.

B. The Rubus Code Generator and Run-Time System

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

Within trigger chains, inter-SWC communication is aggressively optimized to use the most efficient means of communication possible for each communication link. For example, there is no use of semaphores in point-to-point communications within a trigger chain. Another example is sharing of memory buffers between ports when there are no overlapping activation 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.

C. The Rubus Analysis Framework (RAF)

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

supported by the model includes distributed end-to-end response time analysis and shared stack analysis.

D. The Rubus Simulation Model

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

E. Plug-in Framework in Rubus-ICE

The plug-in framework in Rubus-ICE [18] facilitates the implementation of state-of-the-art research results in an isolation (without needing Rubus tools) and their integration as add-on plug-ins (as binaries or source code) with the integrated development environment. A plug-in is interfaced with the builder tool. The plug-ins are executed sequentially which means that the next plug-in can execute only when the previous plug-in has run to completion. Hence, each plug-in reads required attributes as an input, runs to completion and finally writes the results to the Intermediate Compiled Component Model (ICCM) file. An Application Programming Interface (API) defines the services required and provided by a plug-in. Each plug-in specifies the supported system model, required inputs, provided outputs, error handling mechanisms and a user interface. A sequence of main steps in Rubus-ICE, from modeling an application to the generation of code, is depicted in Figure 1. The component-based design of an application is modeled in the Rubus Designer tool. Then the compiler compiles the design model into the ICCM file. After that the builder runs a set of plug-ins sequentially. Finally, a coder tool generates the code.

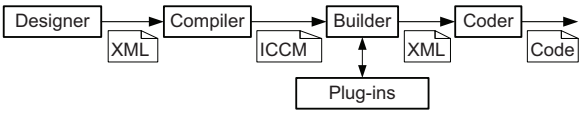

Figure 1. Sequence of steps from design to code generation in Rubus-ICE

IV. Related Work

A. Component Based Development

There exist many component models for the development of distributed systems, e.g., DCOM [19], CORBA [20], EJB [21], etc. These models in their original form are not suitable for the development of resource-constrained DRE systems because they require excessive amount of computing resources, have large memory foot print and have inadequate support for modeling of real-time communication. There are very few commercial component models for the development of DRE systems especially in automotive domain. In the last decade, automotive research community and industry has focused more on the component-based development which led to the development of several component models.

1) AUTOSAR: AUTOSAR (AUTomotive Open System ARchitecture) [6] is a standardized software architecture for the development of software in automotive domain. It can be viewed as a standardized distributed component model [22]. In AUTOSAR, the application software is defined in terms of Software Components (SWCs). The distribution of SWCs, their virtual integration and communication at design time is handled by the Virtual Function Bus (VFB). The run-time representation of VFB for each Electronic Control Unit (ECU) is defined by the Run-Time Environment (RTE). The communication services are provided by the Basic Software (BSW) via RTE to the AUTOSAR SWCs.

When AUTOSAR was being developed, there was no focus placed on the specification and handling of real-time requirements and properties. On the other hand, such requirements and capabilities were strictly taken into account for RCM. As compared to AUTOSAR, RCM clearly distinguishes between the control flow and the data flow among SWCs in a node. AUTOSAR hides the modeling of execution environment. On the other hand, RCM explicitly allows the modeling of execution requirements, e.g., jitter, deadlines, etc., at an abstraction level close to the functional modeling while abstracting the implementation details.

In RCM, special 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-node and inter-node communication at modeling level. Unlike RCM, there are no special components in AUTOSAR for modeling inter-node communication in DRE systems. The Sender Receiver communication mechanism in AUTOSAR is very similar to the pipe-and-filter communication mechanism for component interconnection used in RCM.

2) TIMMO: TIMMO (TIMing MOdel) [15] is an initiative to provide AUTOSAR with a timing model. It describes a predictable methodology and a language, TADL (Timing Augmented Description Language) [23], to express timing requirements and timing constraints during all development phases of automotive embedded systems. TADL is inspired by MARTE (Modeling and Analysis of Real Time and Embedded systems) [24] 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 [25] which is a domain-specific architecture description language used in automotive domain. TIMMO methodology and its model structure abstract the modeling of communication at implementation level of EAST-ADL where AUTOSAR is used. TIMMO and TADL have been evaluated on prototype validators. To the best of our knowledge there is no concrete industrial implementation of TIMMO. Its results will be further validated and brought to the industry in TIMMO-2-USE project [26].

3) ProCom: ProCom [27] is a two-layer component model for the development of distributed embedded systems. At the upper layer, called ProSys, it models a system with concurrent subsystems (active components). At the lower layer, called ProSave, a subsystem is internally modeled in terms of functional components which are passive. ProCom is inspired by RCM. There are a number of similarities between the ProSave modeling layer and RCM, e.g., components are passive, separation between data flow and control flow among components, and pipe-and-filter mechanism for component interconnection. At modeling level, ProCom does not differentiate between inter- and intra-node communication. The support for modeling and holistic timing analysis of DRE systems is a work in progress. Moreover, the development environment and the tools accompanying ProCom are still evolving.

4) COMDES-II: COMDES-II (COMponent-based design of software for Distributed Embedded Systems) [28] 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 (unlike RCM components). At the lower level, the functionality of an actor is modeled in terms of Function Blocks which are passive components similar to the SWCs in RCM. The Operating System (OS) employed by COMDES-II implements fixed-priority timed multitasking scheduling. Whereas, Rubus OS implements hybrid scheduling [29]. The support for development tools and run-time environment in COMDES-II was provided fairly recently [30]. On the other hand, RCM and its tool suite are relatively mature as they are being used in the industry for the development of embedded systems for over 15 years [16].

5) Real-Time CORBA: The middleware technologies such as Real-Time CORBA and minimum CORBA [31] have been used for the development of DRE systems in some projects. Because of higher resource requirements, these technologies may not be suitable for the state-of-the-practice development of resource-constrained DRE systems with hard real-time requirements [32].

B. Response Time Analysis (RTA)

1) RTA of Tasks in a Node: Liu and Layland [33] provided theoretical foundation for analysis of fixed-priority scheduled systems. Joseph and Pandya published the first RTA [34] for the simple task model in [33]. Subsequently, RTA has been applied and extended in a number of ways by the research community. Tindell [35] developed the schedulability analysis for tasks with offsets and it was further extended by Palencia and Gonzalez Harbour [36]. Later, Mäki-Turja and Nolin [37] reduced pessimism from RTA developed in [35], [36] and presented a tighter RTA for tasks with offsets by accurately modeling inter-task interference. RTA [38], [39] has become a powerful, mature and well established schedulability analysis technique. In crux, 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 today [40]. We extract the timing model and implement the analysis in [37] as part of HRTA.

2) RTA of Messages in a Network: There are many real-time network protocols used in DRE systems. In this paper, we will focus only on CAN and some of its high-level protocols. Tindell et al. [41] developed the schedulability analysis of CAN which has served as a basis for many research projects. This analysis has been implemented in the analysis tools that are used in the automotive industry [42], [43]. Later on, this analysis was revisited and revised by Davis et al. [44]. This analysis assumes that all CAN device drivers implement priority-based queues. In [45] Davis et al. pointed out that this assumption may become invalid when some nodes in a CAN network implement FIFO queues and some implement priority-based queues. They extended the schedulability analysis of CAN with FIFO queues.

However, the existing analysis assumes that CAN messages are queued for transmission periodically or sporadically. It does not support the response-times computation of mixed-type CAN messages which are implemented by several high-level protocols for CAN. In order to meet this industrial need, we extend the existing analysis to support mixed messages. The extended analysis can compute the response times of periodic, sporadic and mixed messages in the CAN network where some nodes use FIFO queues while others use priority queues. We also implement both existing and extended analysis as part of HRTA.

3) Holistic RTA (HRTA): It combines the analysis of nodes and a network. Hence, it computes the response times of event chains that are distributed over several nodes in a DRE system. In this paper, we consider the timing model that corresponds to the holistic schedulability analysis for DRE systems [7]. In [46], Pop et al. provide a holistic schedulability analysis of distributed embedded systems in which the tasks are both time- and event-triggered. The analysis is developed for ST/DYN protocol bus that uses static and dynamic phases for sending messages. As compared to this approach, we use CAN protocol for network communication and the analysis supports any combination of periodic, sporadic and mixed messages.

V. Paper Contributions

The contributions in this paper are organized in four parts. In the first part, we introduce a new technique for modeling legacy network communication in DRE systems. In the second part, we present a method to extract the end-to-end timing models from component-based DRE systems. In the third part, we identify a need for the extension of

existing response-time analysis of CAN, and accordingly, we present the extended analysis. Finally, in the fourth part, we provide a proof-of-concept implementation of the techniques developed in previous three parts. In this Section, we provide the summary of these contribution. The details of these four contributions are discussed in [47], [48], ([49], [50]), and [51] respectively.

A. Modeling of Legacy Network Communication in Component-based DRE Systems

This contribution addresses first research question. We introduce a new approach for modeling real-time network and legacy communication in component-based DRE systems. In order to show usability of our modeling approach, we implement it by extending the existing industrial component model, i.e., RCM. By introducing special-purpose components to encapsulate and abstract the communication protocols in DRE systems, we allow the use of legacy nodes and legacy protocols in a component- and model-based software engineering environment. With the addition of these components, RCM will be able to not only model real-time network communication, but also support state-of-the-practice development of component-based DRE systems.

The proposed extension also allows model- and component-based development of new nodes that are deployed in legacy systems that use predefined communication rules. These extensions also enable adaptation of a node when communication rules change (e.g., due to re-deployment in a new system or due to upgrades in the communication system) without affecting its internal component design. The special-purpose components can be automatically generated from the information about legacy communication or from early design decisions about network communication. Although RCM was selected for the proof-of-concept implementation, the proposed extensions should be generally applicable for the extension of several component models for the development of DRE systems that use the pipe-and-filter style for component interconnection such as ProCom [27] and COMDES-II [28]. This modeling approach is briefly discussed in Section A1 of Appendix A.

B. Extraction of End-to-end Timing Models

This contribution addresses second research question. HRTA is an important activity during the development of DRE systems. In order to perform HRTA of component-based DRE systems, the end-to-end timing models should be extracted from them. The extraction of such models can be challenging because the design and analysis models are usually built using different meta-models. We present a method to extract the end-to-end timing models from component-based DRE systems to facilitate HRTA. This method is built upon the modeling approach that we discussed in the first contribution. We discuss and solve the issues concerning the model extraction such as extraction of unambiguous timing and tracing information from all nodes and networks in the system and tracing of event chains in distributed transactions. The extraction method for end-to-end timing models and the solutions of encountered problems may be applied to several component models that use a pipe-and-filter style for component interconnection. The end-to-end timing model that we considered is also general as it incorporates the analysis of several real-time network protocols used in the automotive domain. To show the applicability of our approach, we demonstrate the implementation of end-to-end timing model extraction in the analysis framework of the existing industrial tool suite Rubus-ICE. The solution for tracing event chains in distributed transactions and the extraction method for end-to-end timing models is briefly discussed in Sections A2 and A3 of Appendix A.

C. Extension of the Existing Analysis for CAN

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

In order to support the development of DRE systems employing high-level protocols for CAN, there is a need to extend the existing analysis. We extend the existing response-time analysis of CAN to support mixed messages. The extended analysis is generally applicable to any high-level protocol for CAN that uses periodic, sporadic, and both periodic and sporadic transmission of messages. However, the extended analysis assumes that all nodes connected to the network implement priority queues. We further extend this analysis by building it upon the analysis for CAN with FIFO queues [45]. Hence, the extended analysis supports the worst-case response-time computation of mixed messages in the CAN network where some nodes use FIFO queues while others use priority queues.

D. Proof-of-Concept Implementation

In this contribution we validate our solutions. In order to transfer the new modeling techniques and extended analysis to the industrial tools we need to validate them first. We found out that the process of implementing and integrating state-of-the-art real-time analysis with an existing industrial

tool suite offers many challenges. The Implementer has to not only code and implement the analysis in the tool suite, but also deal with several other issues. We present the implementation of HRTA as a plug-in for the existing industrial tool suite Rubus-ICE. As part of HRTA, we implemented the existing as well as the extended analysis discussed in the third contribution. The implemented HRTA is general as it supports the integration of response-time analysis of various networks without a need for changing the holistic algorithm. The implementation of HRTA in Rubus-ICE is briefly discussed in Appendix A. To the best of our knowledge, Rubus-ICE is the only tool suite that implements RTA of mixed messages in CAN [49] and a tighter version of offset-based RTA algorithm [37] as part of HRTA.

We discuss and solve encountered issues and highlight gained experiences during the implementation, integration and evaluation of HRTA plug-in. We believe that most of the experiences gained and solutions to the issues encountered in this work maybe applicable when other complex real-time analysis techniques are implemented in any industrial tool suite that supports a plug-in framework (for the integration of new tools) and component-based development of DRE systems. Finally, we provide a proof of concept for all modeling approaches and extended analysis discussed in this paper by modeling an automotive industrial application (autonomous cruise control system) using extended RCM and analyzing it with HRTA plug-in in Rubus-ICE.

E. Discussion

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

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

F. Industrial Impact of Contributions

The new modeling techniques and extended analysis that we introduced in this paper are incorporated in the new release of RCM and Rubus-ICE (Version 4.0).

VI. Conclusion

A. Summary and Conclusion

In this paper we introduced new techniques to provide a model- and component-based support for communications-oriented development of DRE control systems. We proposed a new approach for modeling legacy network communication in component-based DRE systems. The proposed approach abstracts the implementation and configuration of communications in DRE systems. It enables the communication capabilities of a node very explicit, but efficiently hides the implementation or protocol details. Moreover, 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 component design.

We also presented a method to extract end-to-end timing models from component-based DRE systems that are developed using above modeling approach. The purpose of extracting the end-to-end timing models is to support HRTA of DRE systems. We believe, these techniques may be suitable for several other component models for DRE systems that use a pipe-and-filter style for component interconnection. Moreover, these techniques can be used for any type of "inter-model signaling", where a signal leaves one model (e.g., a node, or a core, or a process) and appears again in some other model.

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

We provided a proof-of-concept implementation of our modeling and analysis approaches by extending the existing industrial component model, i.e., RCM; implementing the extended HRTA in an industrial tool suite, i.e., Rubus-ICE; and conducting an automotive-application case study. The extended HRTA that we implemented in Rubus-ICE is general as it supports the integration of real-time analysis of various networks without a need of changing the holistic algorithm. The analysis engines that we provide are able to predict important execution characteristics of the system such as holistic response times without a need for tedious and expansive testing. The proposed solutions and gained experiences in this work should provide guidance for the implementation of other complex real-time analysis techniques in any other industrial tool suite that supports a plug-in framework (for the integration of new tools) and component-based development of DRE systems.

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

B. Future Work

An interesting future research direction is to investigate and develop patterns that allow transformation between several domain-specific modeling languages in the vehicular domain. The idea is to bridge the semantic gap between functional models (expressed in standard languages as EAST-ADL [25] and/or proprietary languages such as Simulink [52] or Statemate [53]) and execution models (expressed in standard languages like TADL [23] and Autosar [6] and/or proprietary languages like RCM). It would also be interesting and useful to facilitate the exchange of analysis models and tools between RCM and several other component models and tools used for the development of automotive embedded systems.

In the future, the HRTA plug-in can be expanded by implementing and integrating the analysis of other network communication protocols (e.g., Flexray, switched ethernet, etc.) within the holistic analysis algorithms discussed in this paper. Another future work could be providing a support for asynchronous data-flow using the two different semantics of data-age and reaction (described in [54]) in Rubus-ICE.

ACKNOWLEDGEMENT

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

REFERENCES

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

[2] M. Barr, "Embedded Systems Glossary." http://www.netrino.com/Embedded-Systems/Glossary.

[3] R. N. Charette, "This Car Runs on Code," *Spectrum, IEEE*, vol. 46, no. 2, 2009, http://spectrum.ieee.org/green-tech/advanced-cars/this-car-runs-on-code.

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

[5] I. Crnkovic and M. Larsson, *Building Reliable Component-Based Software Systems*. Norwood, MA, USA: Artech House, Inc., 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] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocess. Microprogram.*, vol. 40, pp. 117–134, April 1994.

[8] R. B. GmbH, "CAN Specification Version 2.0," postfach 30 02 40, D-70442 Stuttgart, 1991.

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

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

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

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

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

[14] "MilCAN," http://www.milcan.org/.

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

[16] "Arcticus Systems," http://www.arcticus-systems.com.

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

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

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

[20] "OMG, Common Object Request Broker Architecture (CORBA) , Version 3.1," January 2008. [Online]. Available: http://www.omg.org/spec/CORBA/3.1

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

[22] H. Heinecke et al., "AUTOSAR – Current results and preparations for exploitation," in *Proceedings of the 7th Euroforum Conference*, ser. EUROFORUM '06, May 2006.

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

[24] "The UML Profile for MARTE: Modeling and Analysis of Real-Time and Embedded Systems," January 2010. [Online]. Available: http://www.omgmarte.org/

[25] "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.

[26] "TIMMO-2-USE," http://www.timmo-2-use.org/.

[27] S. Sentilles, A. Vulgarakis, T. Bures, J. Carlson, and I. Crnkovic, "A Component Model for Control-Intensive Distributed Embedded Systems," in *Proceedings of the 11th International Symposium on Component Based Software Engineering (CBSE2008)*. Springer Berlin, October 2008, pp. 310–317.

[28] X. 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*, August 2007, pp. 199 –208.

[29] J. Mäki-Turja, K. Hänninen, and M. Nolin, "Efficient Development of Real-Time Systems Using Hybrid Scheduling," in *International Conference on Embedded Systems and Applications*, June 2005.

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

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

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

[33] C. Liu and J. Layland, "Scheduling algorithms for multi-programming in a hard-real-time environment," *ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[34] M. Joseph and P. Pandya, "Finding Response Times in a Real-Time System," *The Computer Journal (British Computer Society)*, vol. 29, no. 5, pp. 390–395, October 1986.

[35] K. W. Tindell, "Using offset information to analyse static priority pre-emptively scheduled task sets," Dept. of Computer Science, University of York, Tech. Rep. YCS 182, 1992.

[36] J. Palencia and M. G. Harbour, "Schedulability Analysis for Tasks with Static and Dynamic Offsets," *Real-Time Systems Symposium, IEEE International*, p. 26, 1998.

[37] J. Mäki-Turja, , and M. Nolin, "Tighter response-times for tasks with offsets," in *Real-time and Embedded Computing Systems and Applications Conference (RTCSA)*. Springer-Verlag, August 2004.

[38] N. Audsley, A. Burns, R. Davis, K. Tindell, and A. Wellings, "Fixed priority pre-emptive scheduling:an historic perspective," *Real-Time Systems*, vol. 8, no. 2/3, pp. 173–198, 1995.

[39] L. Sha, T. Abdelzaher, K.-E. A. rzén, 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*, vol. 28, no. 2/3, pp. 101–155, 2004.

[40] M. Nolin, J. Mäki-Turja, and K. Hänninen, "Achieving Industrial Strength Timing Predictions of Embedded System Behavior," in *ESA*, 2008, pp. 173–178.

[41] K. Tindell, H. Hansson, and A. Wellings, "Analysing real-time communications: controller area network (CAN)," in *Real-Time Systems Symposium (RTSS) 1994*, pp. 259 –263.

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

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

[44] R. Davis, A. Burns, R. Bril, and J. Lukkien, "Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised," *Real-Time Systems*, vol. 35, pp. 239–272, 2007.

[45] R. I. Davis, S. Kollmann, V. Pollex, and F. Slomka, "Controller Area Network (CAN) Schedulability Analysis with FIFO queues," in *23rd Euromicro Conference on Real-Time Systems (ECRTS11)*, July 2011.

[46] T. Pop, P. Eles, and Z. Peng, "Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems," in *Proceedings of the tenth international symposium on Hardware/software codesign*, ser. CODES '02. New York, NY, USA: ACM, 2002, pp. 187–192. [Online]. Available: http://doi.acm.org/10.1145/774789.774828

[47] S. Mubeen, J. Mäki-Turja, M. Sjödin, and J. Carlson, "Analyzable Modeling of Legacy Communication in Component-Based Distributed Embedded Systems," in *37th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), 2011*, Sep. 2011, pp. 229 –238.

[48] S. Mubeen, J. Mäki-Turja, and M. 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*. Springer, October 2011, pp. 1–6.

[49] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "Extending schedulability analysis of controller area network (CAN) for mixed (periodic/sporadic) messages," in *Emerging Technologies Factory Automation (ETFA), IEEE 16th Conference on*, sept. 2011.

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

[51] S. Mubeen, J. Mäki-Turja, and M. 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)*, April 2012.

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

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

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

APPENDIX A

A1. Support for Modeling of Legacy Communication

In order to model legacy network communication in DRE systems, we introduced special-purpose components, i.e., Network Specification (NS), Output Software Circuit (OSWC) and Input Software Circuit (ISWC) in [47].

1) Network Specification: It represents the model of communication in a physical network. There is one NS for each network protocol. The protocol-independent part of NS defines messages, data-elements mapped to these messages, message properties, i.e., a message ID, a unique sender node ID, a list of receiver nodes IDs and an ordered set of signals. The protocol-dependent part of NS defines the behavior semantics of each message according to the protocol used for network communication. It contains complete information of all the frames which are sent on the bus. Moreover, it contains a Signal Mapping object that defines all the rules concerning the following questions. How are signals packed into the frames? How many signals a message contains? How are signals encoded into the frames at the sender node? How are signals decoded from the received frames and sent to the respective SWCs at the receiver node?

2) Output and Input Software Circuits: The OSWC component is the model representation of signals in an outgoing message to the network. OSWC has only one trigger in-port and at least one data in-port. Each data in-port is associated with one signal in NS. OSWC has no data and trigger out-ports. It uses protocol-specific rules, specified in the protocol-specific part of NS, while encoding data and mapping signals to a frame. In this way, OSWC provides a clear abstraction to the SWCs that send signals to one of its data in-ports. Thus, SWCs are kept unaware of the protocol-specific details such as signal-to-frame mapping, data type encoding and transmission patterns of frames.

The ISWC component is the model representation of signals in an incoming message from the network. It has one unconditional trigger out-port. An unconditional trigger port produces a trigger signal every time the SWC is

executed. There is at least one data out-port in the ISWC component. Each data out-port is associated with one signal in NS. ISWC has no data out-ports. There is one trigger in-port in every ISWC component which is triggered when a frame arrives from the network. When a frame arrives at a node, the physical network drivers and protocol-specific implementation of ISWC extract the signals (zero or more signals per frame). 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 component (the tracing information is provided in NS), and the corresponding trigger port is triggered. The model representation of OSWC, ISWC and NS in a two-node DRE system is shown in Figure 2.

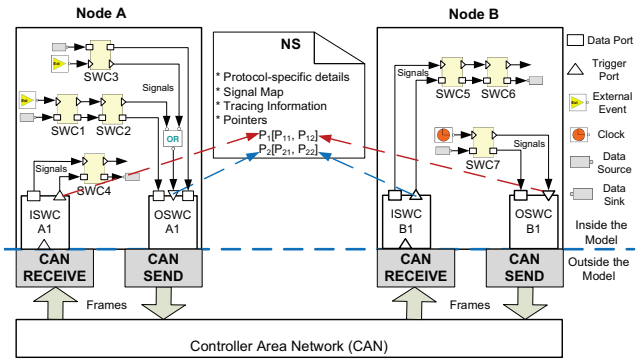

Figure 2. Model of OSWC, ISWC and NS in a two-node DRE system

A2. Tracing of Event Chains in Distributed Transactions

The tracing information of all event chains in the modeled DRE application is provided in NS. We assign pointers (references) to the input trigger ports of all OSWCs and the output trigger ports of all ISWCs along the same distributed transaction. All such pointers for all the event chains in the system are specified in NS. Consider again the model of a two-node DRE system shown in Figure. 2. Both the nodes send messages to each other. There is a pointer array P_1 in NS that references the trigger in-port of OSWC B1 in Node B and trigger out-port of ISWC A1 in node A. Similarly, a pointer array P_2 is stored in NS that points to the trigger in-port of OSWC A1 in Node A and trigger out-port of ISWC B1 in node B. In this way, the event chains in distributed transactions in a DRE system can be traced.

A3. Extraction of End-to-End Timing Model in Rubus-ICE

In Rubus-ICE, a DRE application is modeled in Rubus Designer. It is then compiled to ICCM file which contains the compiled component model and timing and tracing information of the modeled system. The end-to-end timing model that is implemented in RAF, extracts the required timing and tracing information from ICCM file as shown in Figure 3. From the extracted timing model, RAF performs the end-to-end timing analysis and then provides the results, i.e., response times of individual tasks, response times of network messages, end-to-end response times of event chains, network utilization, etc., back to Rubus-ICE.

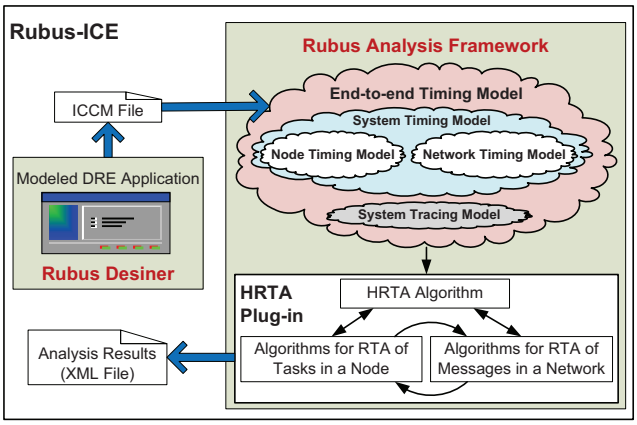

Figure 3. Extraction of end-to-end timing model and conceptual view of HRTA plug-in in Rubus-ICE

A4. Implementation of Extended HRTA in Rubus-ICE

The conceptual view of HRTA that is implemented as a plug-in in Rubus-ICE is shown in Figure 3. The HRTA algorithm iteratively uses the algorithms for node and network analysis. We implemented a tighter version of the offset-based RTA algorithm [37] to compute the response-times of tasks in nodes. In order to analyze network messages, we implemented a general RTA of CAN (it supports several high-level protocols for CAN). It is based on the existing analysis [41], [44] as well as the extended analysis of CAN for mixed messages [49] as discussed in the third contribution. The pseudocode of HRTA algorithm is shown in Algorithm 1. The details about implementation issues, solutions and implemented analysis are described in [51].

Algorithm 1 HRTA Algorithm

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