Modeling of Legacy Communication in Component-Based Distributed Embedded Systems

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

Abstract

We propose the addition of special purpose component types to a commercially existing component model, the Rubus Component Model (RCM). The purpose of the new component types is to encapsulate and abstract the communications protocol and configuration in a component based and model based software engineering setting. With the addition of these new component types, RCM will be able to support state-of-the-practice development processes of distributed embedded systems where communication rules are defined early in the development process. We also show how an end-to-end timing model can be extracted from a distributed embedded system, modeled with RCM, to perform end-to-end timing analysis.

Keywords *Model-Based Software Engineering; Component-Based Software Engineering; Distributed embedded systems.*

I. Introduction

With the recent advancement in technology, embedded systems have become more and more complex. In order to deal with this complexity, lower development cost, reduce time-to-market and time-to-test, allow reusability and support modeling at higher level of abstraction, etc., the research community proposed the employment of Model-Based Engineering (MBE) and Component-Based Software Engineering (CBSE) for the development of embedded systems [1] [2].

Software development of distributed embedded systems is more complex as compared to single processor embedded systems. When MBE and CBD are used for the development of resource constrained and hard real-time distributed embedded systems, modeling of communication infrastructure arises as another challenge. The component model for the development of such systems should not only be resource efficient but it should also abstract the application software from the communication infrastructure. Moreover, it should also be able to model the legacy (previously developed) communications and legacy systems.

In this paper we propose the extension of a commercially existing component model, the Rubus Component Model (RCM) [3], by adding special purpose component types to it. RCM is a component model used for the development of resource constraint real-time embedded systems. It supports glue-code generation, end-to-end delay analysis, and resource requirements estimations. The purpose of the new component types, introduced in RCM, is to encapsulate and abstract the communications protocol and configuration in a component based and model based software engineering setting.

Our main goals in introducing these components are:

1) Allow model-based and component-based development of new nodes that are deployed in legacy systems that use predefined communications rules.
2) Support state-of-the-practice development processes where communications rules are defined early in the development process.
3) Enable adaptation of a node when communications rule change (e.g. due to re-deployment in a new system or due to upgrades in the communication system) without affecting the internal component design.

These goals are to be realized in RCM. The scope of this paper is PSMs (Platform Specific Models) for distributed embedded systems. With PSM we mean that the software component has been allocated to nodes and any adaptation to specific node characteristics (e.g. device drivers and memory layouts) has been added to the model.

Using our new components, nodes can be developed without explicit knowledge about the communication configuration.

Paper Layout

The rest of the paper is organized as follows. Section II presents the Rubus concept, the component model and its development environment. In section III, we present the related research and compare different modeling

approaches with ours. In section IV, we describe the new modeling objects to support modeling of legacy communication. Section V illustrates the extraction of timing model, from distributed embedded systems modeled with RCM, for end-to-end timing analysis. Section VI concludes the paper and section VII presents the future work.

II. Background – The Rubus Concept

The Rubus concept is based around the Rubus Component Model (RCM) [3] and its development environment Rubus-ICE (Integrated Component Environment) [4], 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)

The purpose of the component model is to express the infrastructure for software functions i.e. the interaction between the software functions in terms of data and control flow. One important principle is to separate code and infrastructure, i.e., explicit synchronization or data access should all be visible at the modeling level. In RCM, the basic component is called Software Circuit (SWC). By separating code and infrastructure RCM facilitates analysis and reuse of components in different contexts (an SWC has no knowledge how it connects to other components).

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

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

An example system in RCM is shown in figure1. In this figure one can see how components interact with external events and actuators with regard to both data and triggering. Triggering events can consist of interrupts, internal periodic clocks, and internal and external events. Furthermore, the component model has the possibility to encapsulate SWCs into software assemblies enabling the designer to construct the system at different levels of abstraction.

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

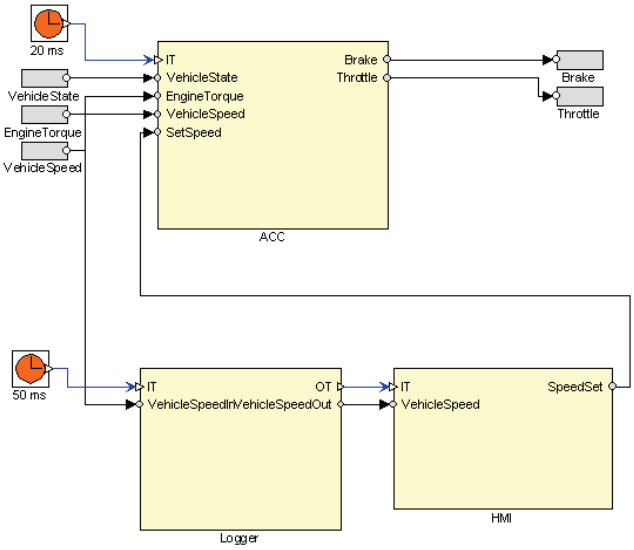

Fig. 1. An example system in RCM

SWC communication is aggressively optimized to use the most efficient means of communication possible for each communication link (e.g. no use of semaphores in point-to-point communications within a chain, and 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

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

III. Related Research

There exist many component models for the development of distributed systems e.g. Distributed Component Object Model (DCOM) [5], Common Object Request Broker Architecture (CORBA) [6], Enterprise JavaBeans (EJB) [7] etc. These component models in their original form are not suitable for the development of resource constrained distributed embedded systems with hard real-time requirements because they require excessive amount

of computing resources, have large memory foot print and have inadequate support for modeling of real-time communication. There are very few commercially existing component models for the development of distributed embedded and real-time systems especially in automotive domain. In the last decade, automotive research community and industry has focused more on the component based development of automotive embedded systems which led to the development of various solutions, approaches, methodologies, and models.

AUTOSAR (AUTomotive Open System ARchitecture) [8] is a standardized software architecture for the development of software in automotive domain. It can be viewed as a standardized distributed component model [9]. 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 Virtual Function Bus (VFB). Run-Time representation of VFB for each Electronic Control Unit (ECU) is defined by 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, the main objective was to build a standardized infrastructure for automotive software development while handling of timing related information during the development was not considered. Furthermore, no focus was laid on specifying and handling of timing properties and real-time requirements during the process of system development. On the other hand, such requirements and capabilities were strictly taken into account right from the beginning during the development of RCM. AUTOSAR describes embedded software development at relatively higher level of abstraction as compared to RCM. A Software Circuit in RCM resembles more to a runnable entity which is the schedulable part of AUTOSAR SWC. As compared to AUTOSAR, RCM clearly distinguishes between the control flow and the data flow among SWCs in a node. AUTOSAR hides the modeling of execution environment whereas RCM explicitly highlights it.

In RCM, special interface objects (NOI and NII), which we will introduce in the next section, are used if SWCs require inter-ECU communication otherwise SWCs communicate via data and trigger ports. On the other hand, AUTOSAR does not differentiate between intra-node and inter-node communication at modeling level. Unlike RCM, there are no special components in AUTOSAR for inter-node communication. AUTOSAR SWCs use interfaces for all types of communications which can be of two types, i.e. Sender-Receiver and Client-Server. The Sender-Receiver communication mechanism in AUTOSAR is very similar to the pipe-and-filter communication mechanism used in RCM.

TIMMO (TIMing MOdel) [10] describes a predictable methodology and a language, TADL (Timing Augmented Description Language) [11], to express timing requirements and timing constraints during all design phases in the development of automotive embedded systems. TIMMO development methodology makes use of structural modeling provided by EAST-ADL [12] which is a standard architecture description language to model a system at various levels of abstraction. The model structure of TIMMO abstracts the modeling of communication at implementation level (defined by EAST-ADL) where the methodology proposes to use AUTOSAR. Both TIMMO methodology and TADL have been evaluated on prototype validators. To the best of our knowledge there is no concrete industrial implementation of TIMMO. In TIMMO-2-USE project [13], the results of TIMMO will be further validated and brought to the industry.

Object Management Group (OMG) defined middleware technology such as Real-Time CORBA, minimum CORBA and CORBA lightweight services for the development of real-time and distributed embedded systems [14]. Real-Time CORBA has been used to develop distributed embedded systems [15] [16]. Because of higher resource requirements, these models may not be suitable for the development of resource constrained distributed embedded systems with hard real-time requirements.

COMDES-II (COMponent-based design of software for Distributed Embedded Systems) provides a component-based framework for the development of distributed embedded control systems [17]. It uses labeled (named) messages for the network communication. The scheduling policy used by OS in COMDES-II is fixed priority timed multitasking scheduling. On the other hand, Rubus Operating system implements hybrid, static and dynamic, scheduling without using timed multitasking [18].

IV. Support for Legacy Communication

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

To support abstraction of the implementation of communications in a node, we propose the introduction of two special purpose modeling elements: the Network Input Interface (NII) and the Network Output Interface (NOI). In order to represent the model of communication in a physical bus, we propose another modeling object called Network Specification (NS). In this section we describe these three new modeling objects in detail.

A. Network Specification (NS)

It is the model representation of a physical bus. There are two parts of NS. One is protocol independent and the other is protocol dependent. The protocol independent part defines a message and its properties such as ID, sender node ID, list of receiver nodes IDs,

list of RCM signals etc. The protocol dependent part is specific to each protocol i.e. it will be different for HCAN, CANopen, Flexray etc. There is one NS per bus. The network dependent part of NS contains complete information of all the frames which are sent on the bus. It also describes the properties of frames.

In RCM, a frame is a collection of RCM signals. A signal has a name, data type, resolution, real-time properties etc. The frame properties described by NS include identifier, priority, transmission type, sender node ID, list of receiver node IDs, whether a frame is an IN frame or an OUT frame, real-time requirements etc. Transmission type of a frame can be periodic, event or mixed (transmitted periodically as well as on arrival of an event).

The components inside a single node communicate with each other using data and control signals separately. However, if a component on one node intends to communicate with a component on another node via a network (bus) then the signals are packed into frames. These frames are then transmitted over the network. Here, some questions arise regarding the network communication. How the signals are packed into frames? How the signals are encoded into the frames at the sender node? How the signals are decoded from the frames and sent to the respective SWCs at the receiver node? All the rules that are concerned with the answers to these questions are specified in the Signal Mapping. Apart from defining the frame properties, NS also defines the Signal Mapping.

B. Network Input Interface (NII) Component

It is the model representation of incoming signals from the network. Thus, NII component describes all signals that can be received by a node from a network. There is one NII component per network that the node is connected to. Associated to each signal is a data-port and a trigger-port. When a frame arrives at the node, the physical bus driver and protocol specific implementation of the NII 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 corresponding data-port and the trigger-port is triggered. If the trigger is not used then it can be connected to the ground. Figure 2 graphically illustrates the NII component.

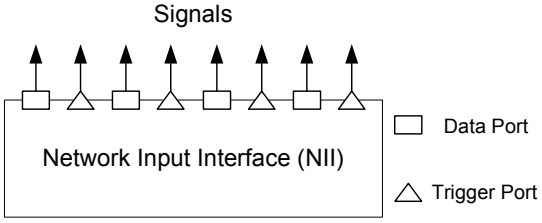

Fig. 2. Graphical illustration of NII

C. Network Output Interface (NOI) Component

The NOI is a component that describes all signals which can be sent from a node on a network. Hence, it is the model representation of the outgoing signals on the network. There is one NOI per network that the node is connected to. The major difference from the NII is that the NOI does not have any trigger ports. Conceptually, the NOI has an implicit trigger port for each data port– however, to lessen the burden for the modeler these ports are omitted from the model. The NII uses protocol specific rules on how to map signals to frames and encode data in the frames. The NII also uses protocol specific rules to decide when to send each frame. Thus, the SWCs that use the NII are kept unaware about details such as signal-to-frame mapping, data-type encoding, and transmission patterns (common transmission patterns are: periodic, on-change, on-change with minimum distance between messages, etc.). Figure 3 graphically illustrates the NOI component.

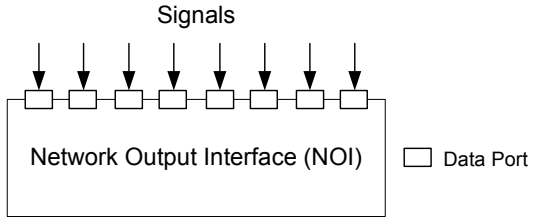

Fig. 3. Graphical illustration of NOI

Both NII and NOI components can be automatically generated from NS by a Network Configuration Tool. This tool also carries out mapping between NS and network interface components and vice versa. The network interface components are translated into a set of SWCs to execute the protocol at run-time. Figure 4 graphically illustrates the model of network communication with new modeling components in RCM.

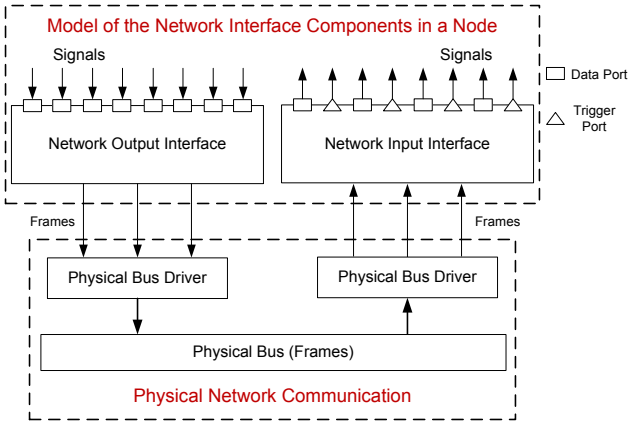

Fig. 4. Model of NOI and NII in a node

Analyzability was one important aspect that was kept in mind while introducing the new component types in

RCM. The objective was to enable RCM to not only model the legacy communication but also to analyze the end-to-end timing behavior of the modeled system. In the next section we will discuss how the required timing information is extracted from a distributed embedded system, modeled with RCM, to perform end-to-end timing analysis.

V. Extraction of Timing Model for End-to-End Timing Analysis

In real-time systems, the time at which the result is available is as important as the correct result. With newly introduced modeling elements in RCM, we can model a complete distributed real-time embedded system. In order to ensure that all timing requirements are met, the modeled system should render itself to end-to-end timing analysis. To perform the timing analysis, end-to-end timing model of the modeled system should be available. The computation model for timing analysis considered in this paper consists of a task model with offsets [19] [20] [21] and a communication model [22]. The task model is used for response-time analysis of tasks in a node whereas, the communication model is used for response time analysis of messages in the network. We illustrate how we can extract end-to-end timing models for distributed transactions modeled with the new modeling elements in RCM. From the extracted model, we will analyze the end-to-end timing for delays and network utilization.

In order to understand which timing information needs to be extracted for end-to-end timing analysis, we consider an example. Figure 5 shows a block diagram of an example distributed embedded system modeled with RCM using the new modeling objects. There are two nodes in the system with three SWCs per node. SWCs communicate with each other using both inter-node and intra-node communication. The inter-node communication takes place via Controller Area Network (CAN) to which the two nodes are connected. An event chain (distributed transaction) that consists of four Software Circuits i.e. SWC1, SWC2, SWC4 and SWC5 is identified with bold lines in figure 5. In this transaction, an event triggers SWC1 which in turn triggers SWC2. SWC2 then sends a signal to NOI which in turn maps it to a CAN frame. This frame is transmitted over CAN bus and it is received by the NII of the receiver node. The NII decodes the signal and places the data on the corresponding data port and it also triggers the corresponding trigger port. The elapsed time between the arrival of the triggering event at the input of SWC1 and the instant when SWC5 gives response is referred to as end-to-end delay of the distributed transaction and it is shown in Figure 5.

The end-to-end timing model should contain timing related information of all transactions in the system. At node level, the timing information includes the total number of tasks in the system, Worst-Case Execution Time

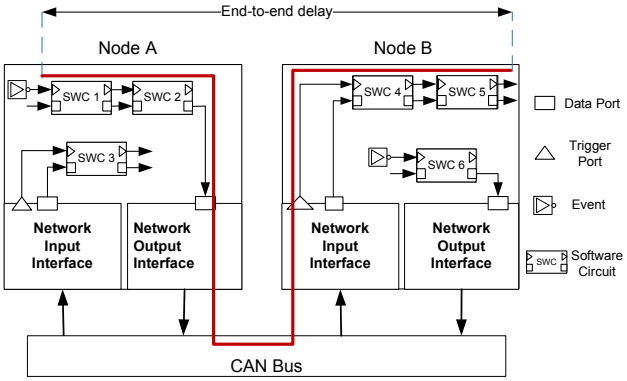

Fig. 5. Example distributed system modeled with extended RCM

(WCET) of each task, trigger Period (periodic activation) or inter-arrival time (event activation) between successive events triggering a chain, Jitter etc. At network level the timing information is specified in NS . It includes bus speed, number of frames, frame transmission time, frame period (periodic frame) or inter-arrival time (event frame) or both (mixed-type frame), frame jitter etc.

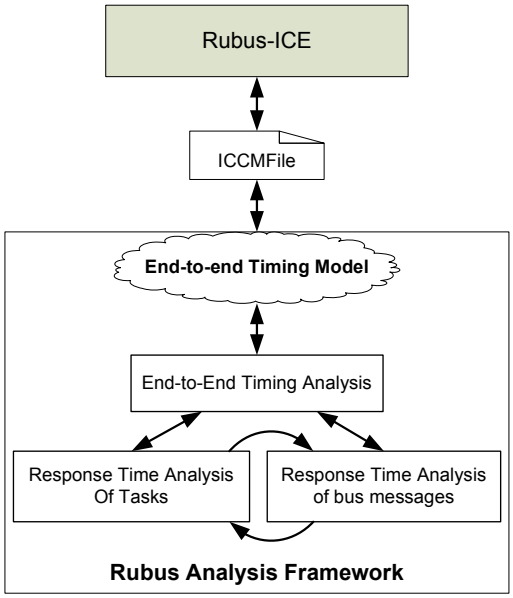

Fig. 6. Extraction of end-to-end timing model for timing analysis in Rubus-ICE

In Rubus-ICE, when the designed model is completed it is compiled to Intermediate Compiled Component Model (ICCM) file [23]. All the timing information required by the end-to-end timing model is extracted from ICCM file. From this timing model, Rubus analysis framework performs response-time analysis of individual tasks [21], response-time analysis of messages on the network [22] [24] and end-to-end timing analysis [25]. The analysis framework provides the results i.e. response-time of individual tasks, response time of frames, end-

to-end delay, network utilization etc. back to Rubus-ICE. This whole process is depicted in Figure 6.

VI. Conclusion

We introduced new modeling elements in a commercially existing component model for the development of distributed embedded systems. The purpose of the new component types, Network Input Interface and Network Output Interface, is to abstract the implementation and configuration of communications in distributed embedded systems. The components make the communications capabilities of a node very explicit, but efficiently hide the implementation or protocol details. We also demonstrated the extraction of end-to-end timing model from a distributed embedded system modeled with RCM.

VII. Future Work

In future work, the implementation of NII and NOI will be automatically generated from protocol configuration files, like CANopen, DCFs (Device Configuration Files) or for subsets of J1939.

Acknowledgement

This work is supported by Swedish Knowledge Foundation (KKS) within the project EEMDEF. The authors would like to thank the industrial partners Arcticus Systems and Hägglunds BAE Systems for cooperation.

References

[1] 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.

[2] I. Crnkovic and M. Larsson, *Building Reliable Component-Based Software Systems*. Norwood, MA, USA: Artech House, Inc., 2002.

[3] K. Hänninen, J. Mäki-Turja, M. Nolin, M. Lindberg, J. Lundbäck, and K.-L. Lundbäck, "The rubus component model for resource constrained real-time systems," in *3rd IEEE International Symposium on Industrial Embedded Systems*, June 2008.

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

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

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

[7] L. DeMichiel, "Sun microsystems, enterprise javabeans specification, version 2.1," *Sun Microsystems*, pp. 1–635, 2002.

[8] "AUTOSAR Techincal Overview, Version 2.2.2," in *AUTOSAR – AUTomotive Open System ARchitecture, Release 3.1*. The AUTOSAR Consortium, August 2008. [Online]. Available: http://autosar.org

[9] e. a. Harald, "Autosar – current results and preparations for exploitation," in *Proceedings of the 7th Euroforum Conference*, ser. EUROFORUM '06, May 2006.

[10] "TIMMO Methodology , Version 2," in *TIMMO (Timing Model), Deliverable 7*. The TIMMO Consortium, Oct 2009.

[11] "TADL: Timing Augmented Description Language, Version 2," in *TIMMO (Timing Model), Deliverable 6*. The TIMMO Consortium, October 2009.

[12] "EAST-ADL Language," http://www.atesst.org/.

[13] "TIMMO-2-USE," http://www.itea2.org/projects/step/2.

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

[15] 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. Proceedings. International*, 2003, p. 8 pp.

[16] R. Finocchiaro, S. Lankes, and A. Jabs, "Design of a real-time corba event service customised for the can bus," in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, 2004, p. 121.

[17] 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*, 2007, pp. 199 –208.

[18] J. Mäki-Turja, K. Hänninen, and M. Nolin, "Efficient development of real-time systems using hybrid scheduling," in *9th Real-Time in Sweden (RTiS'07)*, August 2007, pp. 157–163.

[19] K. Tindell, "Adding time-offsets to schedulability analysis," Despt. of Computer Science, University of York, England, Tech. Rep., January 1994.

[20] J. Palencia and M. G. Harbour, "Schedulability analysis for tasks with static and dynamic offsets," *Real-Time Systems Symposium, IEEE International*, vol. 0, p. 26, 1998.

[21] J. Mäki-Turja and M. Nolin, "Efficient implementation of tight response-times for tasks with offsets," *Real-Time Syst.*, vol. 40, no. 1, pp. 77–116, 2008.

[22] 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.

[23] K. Hanninen, J. Maki-Turja, S. Sandberg, J. Lundback, M. Lindberg, M. Nolin, and K.-L. Lundback, "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.

[24] 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.

[25] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocess. Microprogram.*, vol. 40, pp. 117–134, April 1994. [Online]. Available: http://dx.doi.org/10.1016/0165-6074(94)90080-9