Extracting End-to-end Timing Models from Component-based Distributed Embedded Systems*

Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin

Abstract In order to facilitate the end-to-end timing analysis, we present a method to extract end-to-end timing models from component-based distributed embedded systems that are developed using the existing industrial component model, Rubus Component Model (RCM). RCM is used for the development of software for vehicular embedded systems by several international companies. We discuss and solve the issues involved during the model extraction such as extraction of timing information from all nodes and networks in the system and linking of trigger and data chains in distributed transactions. We also discuss the implementation of the method for the extraction of end-to-end timing models in the Rubus Analysis Framework.

1 Introduction

The model- and component-based development [1, 2] is often considered a promising choice for the development of distributed embedded systems for many reasons such as handling complexity of embedded software; lowering development cost; reducing time-to-market and time-to-test; allowing reusability; providing flexibility, maintainability and understandability; supporting modeling at higher level of abstraction and timing analysis during the process of system development. In distributed embedded systems with real-time requirements, the timing behavior of the system is as important as its functional behavior. The current trend for the industrial development of such systems, especially in automotive domain, is focused towards handling timing related information and performing timing analysis as early as pos-

Saad Mubeen[†], Jukka Mäki-Turja[†‡] and Mikael Sjödin[†]
[†] Mälardalen University, Sweden. [‡] Arcticus Systems, Sweden.
e-mail: {saad.mubeen, jukka.maki-turja, mikael.sjodin}@mdh.se

* This work is supported by the Swedish Knowledge Foundation (KKS) within the project FEMMVA. The authors thank the industrial partners Arcticus Systems, BAE Systems Hägglunds and Volvo Construction Equipment (VCE), Sweden.

sible during the development process [3, 4, 5]. Hence, the component technology for the development of distributed embedded systems should support the extraction of required timing information into the end-to-end timing model.

Goals and Paper Contribution. Our main goal is to extract the end-to-end timing models from component-based distributed embedded systems that are modeled with the existing industrial component model, i.e., the Rubus Component Model (RCM) [6, 7]. We focus on the following issues.

1. Extraction of timing information from all nodes and networks in a distributed embedded application into the end-to-end timing model.
2. Linking of trigger and data chains in distributed transactions, i.e., chains of tasks that are distributed over more than one node in a distributed embedded system.
3. Implementation of the timing model extraction method in the Rubus Analysis Framework.

Paper Layout. The rest of the paper is organized as follows. In Section 2, we discuss the background and research problem. In Section 3, we discuss main constituents of the end-to-end timing model. In Section 4, we discuss the model extraction method. Section 5 presents the related work. Section 6 concludes the paper.

2 Background and Research Problem

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 [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 [10, 11, 12, 13]. The Rubus concept is based around RCM and its development environment Rubus-ICE (Integrated Component development Environment) [7], which includes modeling tools, code generators, analysis tools and run-time infrastructure. The overall goal of Rubus is to be aggressively resource efficient and to provide means for developing predictable and analyzable control functions in resource-constrained embedded systems.

2.1.1 The Rubus Component Model

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 clocks and events as well as other components. In RCM, the basic component is called Software Circuit (SWC). The execution semantics of an SWC are: upon triggering, read data on data *in-ports*; execute the function; write data on data *out-ports*; and 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.

2.1.2 The Rubus Code Generator and Run-Time System

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

2.1.3 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 shared stack analysis [36] and distributed end-to-end response time and delay analysis [37].

2.1.4 The Rubus Simulation Model

The Rubus SIMulation Model (RSIM) and accompanying tools enable simulation and testing of applications modeled with RCM at various hierarchical levels such as an SWC or 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). To verify the logical functionality of these objects, RSIM supports testing in an automatic generated framework based on the Rubus OS Simulator.

The input data is read from external tools or files, e.g., Matlab, and fed to the simulation process that controls the stimulation of input ports and state variables using probes. The output from the simulation process is fed back to the external tools. By building a simulated environment around the application to be simulated, the execution of the application can be controlled from a high-level tool such as LabView or Matlab/Simulink. The high-level tools control the execution of the simulated target by means of commands to stop and run the target clock a specified number of ticks. The high-level tool sets the input data to the control function to be tested, performs a number of execution steps, and then reads the generated output data. In this way the execution flow can be visualized in each time increment.

2.2 Problem Statement: Linking of Distributed Chains

The distributed transactions in a distributed embedded system may consist of trigger chains, data chains or a combination of both. The first task (component) in a trigger chain is triggered independently, while the rest of the tasks are triggered by their respective predecessors as shown in Fig. 1(a). Whereas, each task in a data chain is triggered independently as shown in Fig. 1(b). A mixed chain is a combination of both trigger and data chains as shown in Fig. 1(c). The end-to-end timing model should include linking and mapping information of all these chains. Moreover, the model should also identify the type of each chain because different timing constraints are specified on different types of chains [37].

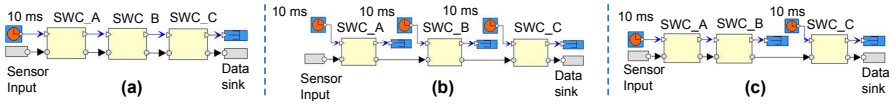

Fig. 1 Example of (a) Trigger chain (b) Data chain (c) Mixed Chain

The linking and mapping problem is common in all types of chains. For simplicity, we consider a distributed embedded system modeled with only trigger chains as shown in Fig. 2. There are two nodes in the system with three SWCs in node A and four SWCs in node B. SWCs communicate with each other by using both inter- 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 (distributed transaction) that is activated by a clock consists of four Software Circuits, i.e., SWC1, SWC2, SWC4 and SWC5. It is identified with a solid-line arrow in Fig. 2. In this transaction, a clock triggers SWC1 which in turn triggers SWC2. SWC2 then sends a signal to the network. This signal is transmitted over the network in a message (frame) and is

received by SWC4 at the receiver node. SWC4 processes it and sends it to SWC5. The elapsed time between the arrival of a triggering event at the input of the task corresponding to SWC1 and the production of response of the task corresponding to SWC5 is referred to as the holistic or end-to-end response time of the distributed transaction and is also identified in Fig. 2. The second trigger chain that is activated by an external event consists of three Software Circuits, i.e., SWC3, SWC6 and SWC7. It is identified by a broken-line arrow in Fig. 2.

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 Fig. 2. In this case, SWC2 communicates with SWC4 by sending signals via the network. Here, the problem is that when a trigger signal is produced by SWC2, it may not be sent straightaway as a message on the network. A message may combine several signals and hence, there may be some waiting time for the signal to be sent on the network. The message may be sent periodically or sporadically or by any other rule defined by the underlying network protocol. When such 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 that generated this message; the *ID* of the destination node; and the *ID*(s) of the task(s) that should receive this message. In order to get a bounded end-to-end delay, a more important question is when and who will trigger the destination SWC when a message is received at node B.

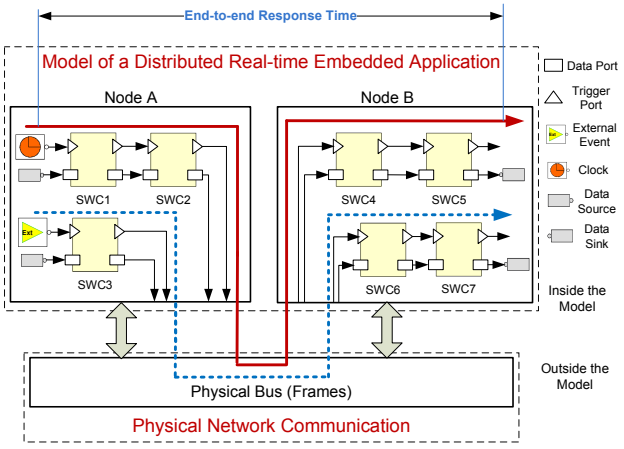

Fig. 2 Trigger chains in distributed transactions

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 in a distributed transaction. Moreover, there is a need to model exit and entry points for RCM models. An exit point is where a message (data) leaves the model and is transmitted according to the protocol-specific rules of the network. Similarly, an entry point is where a message enters the model from the model of the network or any other model. The reason for the need of modeling exit and entry points for RCM models is to get the bounded delays in distributed transactions. The model of entry and exit points will support the use of nodes developed using RCM with the nodes developed by other component technologies.

3 End-to-end Timing Model

The end-to-end timing model consists of timing properties, requirements and dependencies concerning all tasks, messages, task chains and distributed transactions in a distributed embedded system under analysis. Basically, it consists of two models, i.e., system timing model and system linking model. All the required timing information of each node in a distributed embedded application is extracted into a node timing model. Similarly, the timing information of all networks in a distributed embedded application is extracted into a network timing model. Together the node and network timing model comprise the system timing model. All mapping and linking information of distributed chains is extracted into the system linking model.

3.1 System Timing Model

Node Timing Model. The node timing model contains node-level timing information. It is based on a transaction model with offsets developed by [22] and later on, extended by many researchers, e.g., [23, 24]. A node, Γ, consists of a set of k transactions $\Gamma_1, \ldots, \Gamma_k$. Each transaction Γ_i is activated by mutually independent events, i.e., the phasing between them is arbitrary. The activating events can be a periodic sequence of events with a period T_i. In case of sporadic events, T_i denotes the minimum inter-arrival time between two consecutive events.

There are $|\Gamma_i|$ tasks in a transaction Γ_i and each task may not be activated until a certain time, called an *offset*, elapses after the arrival of the external event. By task activation we mean that the task is released for execution. A task is denoted by τ_{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, τ_{ij}, is defined by the following attributes: a priority (P_{ij}), a worst-case execution time (C_{ij}), an offset (O_{ij}), maximum release jitter (J_{ij}), an optional deadline (D_{ij}), maximum blocking time which is the maximum time the task has to wait for a resource that is locked by a lower priority task (B_{ij}). In order to calculate the

blocking time for a task, usually, a resource locking protocol like priority ceiling or immediate inheritance is used. Each task has a worst-case response time denoted by R_{ij}. In this model, there are no restrictions placed on offset, deadline or jitter, i.e., they can each be either smaller, greater or equal to the period.

Network Timing Model. This model contains network-level timing information of a distributed embedded system. A network consists of a number of nodes that are connected through a real-time network. Currently, the model supports Controller Area Network (CAN) and its higher-level protocols such as CANopen, CAN for Military Land Systems domain (MilCAN) and Hägglunds CAN (HCAN) [25]. If a task on one node intends to communicate with a task on another node, it queues a message in the send queue of its node. The network communication protocol ensures the arbitration and transmission of all messages over the network.

Each message m has the following attributes: a unique identifier (ID_m); transmission type showing whether the message is periodic or sporadic or mixed [25]; a unique priority (P_m); transmission time (C_m); release jitter (J_m) which is inherited from the difference between the worst- and best-case response times of the task queueing the message; data payload (s_m) in the message; period (T_m) in the case of periodic transmission, Minimum Update Time (MUT_m) which is the minimum time that should elapse between the transmission of any two sporadic messages in the case of sporadic transmission, or both T_m and MUT_m in the case of mixed transmission [25, 26, 27, 28]; blocking time (B_m) which is the maximum time a message can be blocked by lower priority messages; and worst-case response time (R_m).

3.2 System Linking Model

In distributed embedded systems, the transactions are usually distributed over several nodes. Hence, 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 a network. When there are chains in a distributed embedded 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 (see subsection 2.2) . The extraction of linking information from chains in a distributed real-time system is more complex compared to a single node real-time system.

4 Extraction of End-to-end Timing Model

In this section, we resolve the issues discussed in the previous section. We also show the applicability of our approach by modeling a two-node distributed embedded ap-

plication with RCM. Finally, we present the conceptual organization of the method for the extraction of end-to-end timing models in Rubus-ICE.

4.1 Proposed Solution

4.1.1 Addition of Special Components in RCM

In order to model real-time network communication and legacy communication in distributed embedded systems, we introduced special purpose Software Circuits in RCM, i.e., Output Software Circuit (OSWC) and Input Software Circuit (ISWC) in [14]. There is one OSWC for each message that a node sends to the network. Similarly, there is one ISWC for each message that a node receives from the network. It should be noted that each of these special-purpose components is translated to an individual task at the run time. We also introduced a new object in RCM, i.e., the Network Specification (NS) that represents the model of communication in a physical network [14]. There is one NS for each network protocol. NS contains Signal Mapping which includes the following information: How are signals mapped to messages? How many signals a message contains? How are signals encoded in a message at the sender node? How are signals decoded from a message at the receiving node? The model representation of OSWC, ISWC and NS in a two-node distributed embedded system modeled with RCM is shown in Fig. 3.

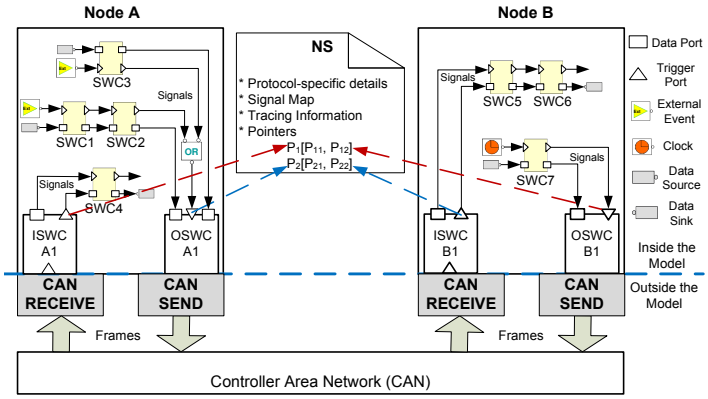

Fig. 3 Example of a two-node distributed embedded system modeled with RCM

When the OSWC component is triggered, it executes the required functionality (e.g., mapping of signals to a message) and then the data (message) is transferred from the RCM model of a node to the network controller or another model of communication network. Therefore, OSWC also represents the model of an exit point for RCM models. Similarly, ISWC component also represents the model of an entry

point for RCM models. Since the trigger *in-ports* of all OSWC components and trigger *out-ports* of all ISWC components along a distributed transaction are referenced in NS, the end-to-end timing delay can be bounded by specifying the delay in the extra-model medium.

4.1.2 Identification of 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 end-to-end timing 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. If this attribute for all tasks (except the first) has value "dependent", the chain will be identified as a trigger chain. On the other hand, if this attribute for more than one task in a chain has "independent" value then the chain will identified as a data chain.

4.1.3 Linking and Mapping of Distributed Chains

The linking information of all distributed chains in the modeled distributed embedded application is provided in the Network Specification. We assign pointers (references) to trigger *in-ports* of OSWCs and the trigger *out-ports* of ISWCs along the same distributed transaction. All such pointers for all trigger chains in the system are specified in the NS.

An example of a two-node distributed embedded system modeled in RCM is shown in Fig. 3. There are four SWCs in Node A while three SWCs in Node B. We consider CAN or any of its high-level protocols for inter-node communication. In this example, the nodes are connected to a CAN network. There are three trigger chains in the system that are distributed over two nodes:

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

The trigger chains EC_1 and EC_2 are triggered by external events whereas the trigger chain EC_3 is triggered by a clock. The references to the trigger ports of OSWC and ISWC in each trigger chain are specified in NS. There is a pointer array P_1 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, all the neighboring components located in different nodes within a distributed trigger chain can be linked to each other. The grey boxes outside the model are specific for each network communication protocol. In this example they represent CAN

SEND and CAN RECEIVE routines. The CAN SEND grey box represents a CAN controller in a node and is responsible for receiving messages from the corresponding OSWC and queueing them for transmission over the network.

When a message arrives at the receiving node, it is transferred by the physical network drivers to the CAN RECEIVE grey box which is responsible for raising an interrupt request and passing the message to the corresponding ISWC component. In this case, the *TrigInterrupt* object in RCM corresponding to the interrupt is connected to the *in-port* of the ISWC component. If CAN drivers use polling-based processing instead of interrupts then the *in-port* of the ISWC component is connected to the clock object in RCM whose period is equal to the polling period. Upon receiving a message, an ISWC component decodes it, extracts signals from it, places the data on the corresponding data port (connected to the data *in-port* of the destination SWC) and triggers the corresponding trigger port by using the linking information in NS. It should be noted that there can be more than one ISWC and OSWC components in a node. It can be seen from Fig. 3 that OSWC and ISWC essentially make the exit and entry points for the node models.

4.2 Extraction of End-to-end Timing Model in Rubus-ICE

In Rubus-ICE, a distributed embedded application is modeled in Rubus Designer. It is then compiled to the Intermediate Compiled Component Model (ICCM). Apart from the compiled component model, ICCM file also includes timing and linking information of the modeled system. The end-to-end timing model that is implemented in the Rubus Analysis Framework, extracts the required timing and linking information from ICCM file as shown in Fig. 4. The end-to-end timing model consists of three models, i.e., node timing model, network timing model and system linking model. From the extracted timing model, the Rubus Analysis Framework performs the end-to-end timing analysis and then provides the results, i.e., response times of individual tasks, response times of network messages, end-to-end response times and delays of distributed chains, network utilization, etc., back to the Rubus-ICE tool suite. The schedulability analysis results of a case study, performed using our timing model extraction method, are presented in [37].

5 Related Work

There are very few commercial component models for the development of distributed embedded systems especially in automotive domain. In our previous work, we carried out a detailed comparison of RCM with various models for distributed embedded systems [14]. We briefly highlight a few of them.

AUTOSAR (AUTomotive Open System ARchitecture) [15] is a standardized software architecture for the development of software in automotive domain. It can

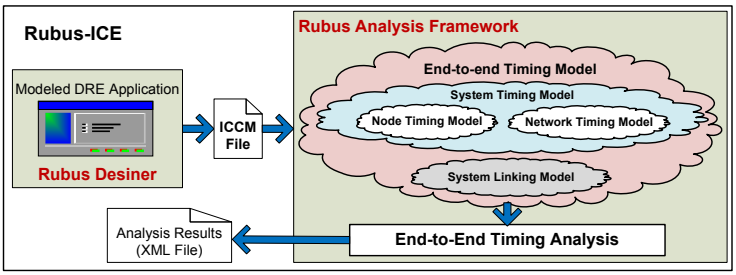

Fig. 4 Extraction of end-to-end timing model in Rubus tool-suite

be viewed as a standardized distributed component model [16]. When AUTOSAR was being developed, there was no focus placed on its ability to specify and handle real-time requirements and properties. 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 a relatively higher level of abstraction compared to RCM. A Software Circuit in RCM more resembles to a runnable entity (a schedulable part of AUTOSAR software component) instead of AUTOSAR software component. As compared to AUTOSAR, RCM clearly distinguishes between control flow and data flow among software components in a node. AUTOSAR hides the modeling of execution environment. Whereas, RCM explicitly allows the modeling of execution requirements, e.g., jitter and deadlines, at an abstraction level close to the functional modeling while abstracting the implementation details.

TIMMO (TIMing MOdel) [5] project 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 [29]. It describes a predictable methodology and a language, TADL (Timing Augmented Description Language) [4], to express timing requirements and timing constraints in all design phases during the development of automotive embedded systems. Both TIMMO methodology and TADL have been evaluated on prototype validators. To the best of our knowledge there is no concrete industrial implementation of TIMMO. In TIMMO-2-USE project [3], TADL2 language has been introduced which includes a major redefinition of TADL. TADL2 also supports the AUTOSAR extensions regarding timing model. Apart from the redefinition of the language, algorithms, tools and a methodology has been developed to model advanced timing at different levels of abstraction. The use cases and validators indicate that the project results are in compliance with the AUTOSAR-based tool chain [29]. Since this project is recently finished, it may take some time for the results of the project to become mature and find their way in the industrial applications.

ProCom [8] is a two-layer component model for the development of distributed embedded systems. ProCom is inspired by RCM, and there are a number of similarities between the ProSave modeling layer (a lower layer in ProCom) and RCM. For example, components in both ProSave and RCM are passive. Similarly, both models

clearly separate data flow from control flow among their components. Moreover, the communication mechanism for component interconnection used in both models is pipe-and-filter. The validation of a complete distributed embedded system, modeled with ProCom, is yet to be done. Moreover, the development environment and the tools accompanying ProCom are still evolving.

BIP framework [30] provides a 3-layered representation, i.e., behavior, interaction and priority for modeling of heterogeneous real-time components. Unlike RCM, it does not distinguish between required and provided or input and output interfaces (or ports). BIP uses triggering, rendezvous and broadcast styles for component interaction. Whereas, RCM used the pipe-and-filter style for interaction among components. BIP provides connections to IF Toolset [31] and PROMETHEUS tools [32] to support modeling, validation, static analysis, model checking and simulation. On the other hand, RCM is supported by the Rubus-ICE tool suite that provides a complete modeling, analysis and simulation support for the component-based development of distributed real-time systems.

Robocop [34, 33] is a component-based framework for the development of middleware in the consumer electronics domain, e.g., consumer devices like mobile phones, DVD players, ATM machines, etc. On the other hand, RCM is intended for the development of real-time embedded systems in the automotive domain. The interaction mechanism among components in both models is also different as Robocop uses request response while Rubus uses pipe and filter. However, there are several similarities between Robocop and RCM, e.g., both have a low resource footprint, both are able to generate C code and both support deployment at the compilation step (in addition, Robocop supports deployment at run-time).

A related research presents a detailed overview of timing aspects during the design activities of automotive embedded systems [18]. In [19], the authors define end-to-end delay semantics and present a formal framework for the calculation of end-to-end delays for register-based multi-rate systems. Like any other timing analysis, they assume that the timing information of the system is available as an input. On the other hand, our focus is on the extraction of required information into an end-to-end timing model to carry out end-to-end timing analysis.

In our previous work, we extended RCM to support modeling and analysis of distributed embedded systems. In [20], we explored various options for modeling of real-time network communication in RCM. In [14], we discussed modeling of legacy communication in component-based distributed embedded systems. We added new components in RCM to encapsulate and abstract the communication protocols, allow use of legacy nodes and legacy protocols in a component- and model-based software engineering environment, and support model- and component-based development of new nodes that are deployed in legacy systems that use predefined communication rules. Further, we highlighted the problem of linking trigger chains in the transactions that are distributed over several nodes in a distributed embedded system [21].

6 Conclusion

In this paper, we discussed the extraction of end-to-end timing models from component-based distributed embedded systems modeled with the industrially available component model, the Rubus Component Model (RCM). The purpose of extracting such models is to perform the end-to-end timing analysis during the development process. We discussed and resolved various issues during the model extraction such as extraction of timing information from all nodes and networks in the system and extraction of linking model containing the linking information of all distributed chains. We also described the implementation of the end-to-end timing model extraction method in the Rubus Analysis Framework.

Although, we discussed the extraction of end-to-end timing models from RCM models, we believe that the model extraction method is also suitable for other component models for the development of distributed embedded systems that use a pipe-and-filter style for component interconnection, e.g., ProCom [8], COMDES [9]. Moreover, our approach can be used for any type of "inter-model signaling", where a signal leaves one model (e.g. a node, or a core, or a process) and appears again in some other model.

References

1. Crnkovic, I. and Larsson, M.: Building Reliable Component-Based Software Systems. Artech House, Inc. (2002)
2. T. A. Henzinger and J. Sifakis. The Embedded Systems Design Challenge. In 14th International Symposium on Formal Methods (FM), Lecture Notes in Computer Science, pages 1-15. Springer, 2006.
3. TIMMO-2-USE. http://www.timmo-2-use.org/.
4. TADL: Timing Augmented Description Language, Version 2. Deliverable 6, October 2009. The TIMMO Consortium.
5. TIMMO Methodology , Version 2. Deliverable 7, October 2009. The TIMMO Consortium.
6. K. Häanninen et.al. The Rubus Component Model for Resource Constrained Real-Time Systems. In 3^{rd} IEEE International Symposium on Industrial Embedded Systems, June 2008.
7. Arcticus Systems. http://www.arcticus-systems.com.
8. S. Sentilles, A. Vulgarakis, T. Bures, J. Carlson, and I. Crnkovic. A Component Model for Control-Intensive Distributed Embedded Systems. In 11^{th} International Symposium on Component Based Software Engineering (CBSE), pages 310-317. Springer Berlin, October 2008.
9. X. Ke, K. Sierszecki, and C. Angelov. COMDES-II: A Component-Based Framework for Generative Development of Distributed Real-Time Control Systems. In 13^{th} IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), pages 199-208, August 2007.
10. BAE Systems Hägglunds, http://www.baesystems.com/hagglunds.
11. Volvo Construction Equipment, http://www.volvoce.com.
12. Mecel, http://www.mecel.se.
13. Knorr-bremse, http://www.knorr-bremse.com.
14. S. Mubeen, J. Mäki-Turja, M. Söodin, and J. Carlson. Analyzable Modeling of Legacy Communication in Component-Based Distributed Embedded Systems. In 37^{th} EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), September 2011.
15. AUTOSAR Techincal Overview, Version 2.2.2. Release 3.1, August 2008.

16. H. Heinecke et al. AUTOSAR Current results and preparations for exploitation. In 7^{th} Euroforum Conference, May 2006.
17. K. Richter et.al. A Timing Verifcation Methodology for AUTOSAR Series Development. In 3^{rd} AUTOSAR Open Conference, 2011.
18. O. Scheickl and M. Rudorfer. Automotive Real Time Development Using a Timing-augmented AUTOSAR Specifcation. In ERTS, 2008.
19. 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 CRTS Workshop, December 2008.
20. S. Mubeen, J. Mäki-Turja, and M. Sjödin. Exploring Options for Modeling of Real-Time Network Communication in an Industrial Component Model for Distributed Embedded Systems. In the Lecture Notes in Electrical Engineering, volume 102, pages 441-458. Springer, 2011.
21. S. Mubeen, J. Mäki-Turja, and M. Sjödin. Tracing event chains for holistic response-time analysis of component-based distributed real-time systems. ACM SIGBED Review, ISSN: 1551-3688, 8(3):48-51, 2011.
22. K. Tindell. Adding Time-Offsets to Schedulability Analysis. Technical report, Department of Computer Science, University of York, England, January 1994.
23. J. Palencia and M. G. Harbour. Schedulability Analysis for Tasks with Static and Dynamic Offsets. In IEEE International Symposium on Real-Time Systems (RTSS), 1998.
24. J. Mäki-Turja and M. Nolin. Efficient implementation of tight response-times for tasks with offsets. Real-Time Systems Journal, 40(1):77-116, 2008.
25. S. Mubeen, J. Mäki-Turja, and M. Sjödin. Extending Schedulability Analysis of Controller Area Network for Mixed (Periodic/Sporadic) Messages. In 16^{th} IEEE Conference on Emerging Technologies and Factory Automation (ETFA), September 2011.
26. S. Mubeen, J. Mäki-Turja, and M. Sjödin. Extending response-time analysis of Controller Area Network (CAN) with FIFO queues for mixed messages. In 16^{th} IEEE Conference on Emerging Technologies and Factory Automation (ETFA), September 2011.
27. 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 9^{th} IEEE International Workshop on Factory Communication Systems (WFCS), May 2012.
28. S. Mubeen, J. Mäki-Turja and M. Sjödin. Worst-Case Response-Time Analysis for Mixed Messages with Offsets in Controller Area Network. In 17^{th} IEEE Conference on Emerging Technologies and Factory Automation (ETFA), September 2012.
29. Mastering Timing Information for Advanced Automotive Systems Engineering. In the TIMMO-2-USE Brochure, September, 2012. Available at: http://www.timmo-2-use.org/pdf/T2UBrochure.pdf.
30. A. Basu, M. Bozga and J. Sifakis. Modeling Heterogeneous Real-time Components in BIP. In the 4^{th} IEEE Conference on Software Engineering and Formal Methods (SEFM), 2006.
31. M. Bozga et. al. The IF Toolset. In the Formal Methods for the Design of Real-Time Systems, Lecture Notes in Computer Science, pp. 237-267, volume 3185, 2004, Springer.
32. Gregor Gssler. Prometheus - A Compositional Modeling Tool for Real-Time Systems. In the Workshop on Real-Time Tools (RT-TOOLS), 2001.
33. J. Muskens, M. R. V. Chaudron, and J. J. Lukkien. A component framework for consumer electronics middleware. In Component-Based Software Development for Embedded Systems. pp 164-184, 2005, Springer-Verlag, Berlin.
34. ROBOCOP project. http://www.hitech-projects.com/euprojects/robocop/deliverables.htm.
35. S. Mubeen, M. Sjödin, J. Mäki-Turja, K-L. Lundbäck and P. Wallin. Automated Model Translations for Vehicular Real-Time Embedded Systems with Preserved Semantics. In ACM SIGBED Review: Special Issue on 33^{rd} IEEE Real-Time Systems Symposium (WIP), 2012.
36. K. Hänninen. Efficient Memory Utilization in Resource Constrained Real-Time Systems. PhD thesis, Mälardalen University, Sweden, 2008.
37. S. Mubeen, J. Mäki-Turja, and M. 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, ISSN: 1361-1384, vol. 10, no. 1, 2013.