Tracing Event Chains for Holistic Response-Time Analysis of Component-Based Distributed Real-Time 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*—In this paper we discuss the problem of tracing event chains (distributed transactions) while extracting an end-to-end timing model from an existing industrial component model, the Rubus Component Model (RCM). RCM supports component-based development of distributed embedded and real-time systems. The purpose of extracting an end-to-end timing model is to perform the holistic response-time analysis of component-based distributed real-time applications modeled with RCM. We present a solution for RCM by introducing special purpose generic components to it. We believe that the solution is also suitable for other component models that use a pipe-and-filter style for component interconnection.**

***Keywords*-holistic response-time analysis; distributed real-time systems; timing model; model-based development.**

I. Introduction

One of the most important requirements during the development of distributed real-time systems is to provide an evidence that each action in the developed system will meet its deadline. Tindell and Clark [1] developed the Holistic Response-Time Analysis (HRTA) to meet this requirement. HRTA is a well established schedulability analysis technique to calculate upper bounds on the response times of event chains (distributed transactions) in a distributed real-time system. In order to perform HRTA, all timing related information of the distributed real-time system under analysis should be available.

The Model- and Component-Based Development [2], [3] is often considered a suitable choice for the development of embedded and real-time systems for many reasons such as: handling complexity of embedded software; lowering development cost; reducing time-to-market and time-to-test; allowing reusability; enabling modeling and analysis at higher level of abstraction, etc. In order to perform HRTA of component-based distributed real-time systems, the component model for the development of such systems should support the extraction of required timing information into an end-to-end timing model.

In this paper, we discuss the extraction of an end-to-end timing model from the industrial component model, the Rubus Component Model (RCM), that is used to develop resource-constrained distributed real-time systems. We discuss the problem of tracing event chains (distributed transactions) while extracting an end-to-end timing model from the modeled application. We also propose a solution to this problem by introducing special purpose components to RCM. We believe that the solution is also suitable for other component models for distributed real-time systems that use a pipe-and-filter style for component interconnection, e.g., ProCom Component Model [4].

The rest of the paper is organized as follows. In Section II, we discuss the holistic response-time analysis. Section III presents the Rubus concept. In Section IV, we discuss the research problem. In Section V, we present a solution to the problem. Section VI summarizes the paper.

II. Holistic Response-Time Analysis

In order to provide an evidence that each action in the system will meet its deadline, *a priori* analysis techniques, also known as schedulability analysis techniques, have been developed by the research community. Response Time Analysis (RTA) is a method to calculate upper bounds on response times of tasks or messages in a real-time system or a network respectively. In [5], it is claimed that amongst the more traditional, analytical, schedulability analysis techniques, RTA of tasks with offsets stands out as the prime candidate because of its better precision and ability to analyze quite complex system behaviors. In this Section, we discuss RTA of tasks in a node (processor), RTA of messages in a network and finally, HRTA.

A. RTA of Tasks in a Node

Liu and Layland [6] provided theoretical foundation for analysis of fixed-priority scheduled systems. Joseph and Pandya published the first RTA [7] for the simple task model presented by Liu and Layland which assumes independent periodic tasks. Subsequently, it has been applied and extended in a number of ways by the research community such as, lifting independent task assumption, analysis of communication networks, analyzing distributed systems, modeling of operating systems overheads, reducing pessimism from traditional RTA, making RTA faster and tighter, etc. Tindell [8] developed the schedulability analysis for tasks with offsets and it was further extended by Palencia and Gonzalez Harbour [9]. RTA [10], [11] 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 [5].

B. RTA of Messages in a Network

There are many protocols such as, CAN (Controller Area Network), TDMA (Time Division Multiple Access), TTCAN (Time-Triggered CAN), FlexRay, etc., that are used for real-time communication in distributed real-time systems. In this paper, we will focus only on the CAN protocol. Tindell et al. [12] developed the schedulability analysis of CAN by adapting the theory of fixed-priority preemptive scheduling for uniprocessor systems. This analysis has served as a basis for many research projects. Moreover, it is implemented in the analysis tools that are used in the automotive industry [13]. Later on, this analysis was revisited and revised by Davis et al. [14].

C. HRTA

HRTA combines the analysis of nodes (uniprocessors) and the network. In other words, it computes the response times of event chains (distributed transactions) that are distributed over several nodes in a distributed real-time system. In this paper, we consider the timing model that corresponds to the holistic schedulability analysis for distributed hard real-time systems that employ CAN protocol for network communication [1]. An example distributed transaction in a distributed real-time system is shown in Figure 1. In this example, the holistic response time is equal to the elapsed time between the arrival of an event (corresponding to the brake pedal input) and the response time of the Task4 (corresponding to the production of a signal for brake actuation).

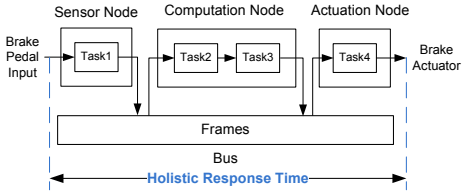

Figure 1. Holistic response-time in a distributed real-time system

III. Background–The Rubus Concept

The Rubus concept is based around the Rubus Component Model [15] and its development environment Rubus-ICE (Integrated Component development Environment) [16], 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 expresses the infrastructure for software functions, i.e., the interaction between the software functions in terms of data and control flow separately. One important principle is to separate functional code and infrastructure implementing the execution model, i.e., explicit synchronization or data access should all be visible at the modeling level. In RCM, the basic component is called Software Circuit (SWC). By separating functional code and the infrastructure RCM facilitates analysis and reuse of components in different contexts (an SWC has no knowledge how it connects to other components). The component model has the possibility to encapsulate SWCs into software assemblies enabling the designer to construct the system at different 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-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.

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 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.

IV. Research Problem

In order to perform the holistic response-time analysis, the end-to-end timing model should be extracted from the distributed real-time system under analysis. The end-to-end timing model should contain timing related information of all transactions and messages in the system. At node level, the timing information includes total number of transactions and the number of tasks in each transaction. Moreover, for every task the Worst-Case Execution Time, offset, maximum release jitter, priority, deadline and period (periodic activation) or inter-arrival time (event activation) between successive events triggering a chain should also be available. At network level, the timing information includes bus speed, total number of messages, message transmission time, size of payload in each message, message period (periodic message) or inter-arrival time (event message) or both (mixed-type message), message jitter, priority of each message, transmission time of each message.

When there are event chains in the system, the timing model should not only contain timing related information but also the tracing information of the event chains. An event chain consists of a number of tasks that are in a sequence and have one common triggering ancestor (e.g., clock, internal and external events, etc.). By tracing information we mean proper sequencing and linking information among all tasks inside the chain. The extraction of tracing information of event chains in a distributed real-time system is more complex compared to a single node real-time system. The component model for the development of real-time systems should provide means to trace event chains and extract related timing information for the purpose of timing analysis. We discuss the issues concerning tracing of event chains in component-based real-time systems in a single node (uniprocessor) as well as in a distributed system.

A. Event Chains in Single-Node Real-Time Systems

Consider an example of an event chain in a single node modeled with RCM as shown in Figure 2. The event chain consists of three Software Circuits, i.e., SWC_A, SWC_B and SWC_C. There is a single activation event (external event trigger) for the chain that triggers SWC_A. When event chains are modeled in a single node in RCM, there are direct triggering connections between every two neighboring SWCs in a chain. For example, SWC_B directly triggers the SWC_C. 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.

RCM is fully capable of modeling and analyzing event chains in single node real-time systems. Hence, there is no problem of tracing event chains (periodic or event).

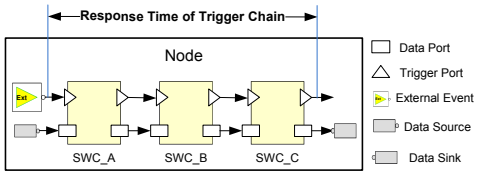

Figure 2. Example of an event chain in a single node

B. Event Chains in Distributed Real-Time Systems

Consider an example of a distributed real-time system modeled with RCM as shown in Figure 3. There are two nodes in the system with three SWCs in node A and four SWCs in node B. SWCs communicate with each other by using both inter-node and intra-node communication. The inter-node communication takes place via a real-time network to which the nodes are connected.

One event chain (distributed transaction) that is activated by a clock consists of four Software Circuits, i.e., SWC1, SWC2, SWC4 and SWC5 and is identified with a solid-line arrow in Figure 3. In this transaction, a clock triggers SWC1 which in turn triggers SWC2. SWC2 then sends a signal to the network. This signal is transmitted over the network in a message (frame) and is received by the SWC4 at the receiver node. This SWC processes it and sends it to SWC5.

The elapsed time between the arrival of a triggering event at the input of the task corresponding to SWC1 and the response of the task corresponding to SWC5 is referred to as the holistic response time of the distributed transaction and is also identified in Figure 3. The second event chain that is activated by an external event consists of three Software Circuits, i.e., SWC3, SWC6 and SWC7. It is identified by a broken-line arrow in Figure 3.

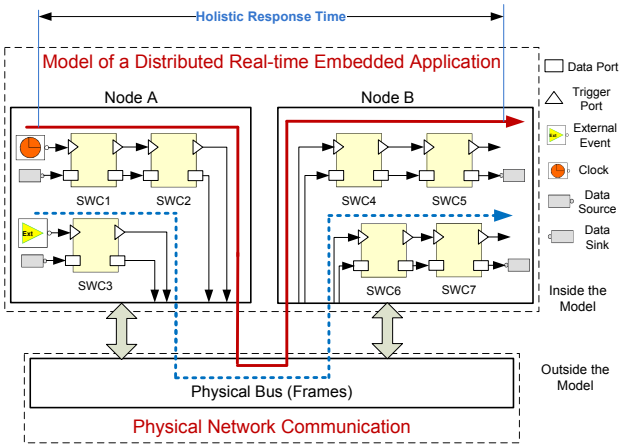

Figure 3. Example of event chains in distributed transactions

There may not be direct triggering connections between any two neighboring SWCs in the chain which is distributed over several nodes, e.g., SWC2 and SWC4 in Figure 3. In this case, SWC2 communicates with SWC4 by sending signals via the network. Here, the problem is that when a trigger signal is produced by SWC2, it may not be sent straightaway as a message on the network. A message may combine several signals and hence, there may be some waiting time for the signal to be sent on the network. The message may be sent periodically or sporadically or by any other rule defined by the underlying network protocol.

When such event chains (distributed transactions) are modeled with a component model, it is not straightforward to trace the event chains to extract the timing model. For example, if a message is received at node B then the following information should be available to correctly link the received message in a chain: the ID of the sender node; the ID of the task that generated this message; the ID of the destination node; the ID(s) of the task(s) that should receive this message, etc.

Discussion

The existing modeling components in RCM do not provide enough support to trace the event chains (distributed over several nodes) and extract corresponding timing information. Therefore, there is a need to introduce special modeling objects in the component model to provide the tracing information of event chains to extract timing information for the timing model. Moreover, there is a need to model exit and entry points in the component model. An exit point is where a message (data) leaves the model and is transmitted according to the protocol-specific rules of the network. Similarly, an entry point is where a message enters the model from the network.

Although, we discussed this problem for RCM, we believe that this problem may occur during the development of any other component model for distributed real-time systems that uses a pipe-and-filter communication mechanism for component interconnection. The problem may also exist in any type of "inter-model signaling", where a signal leaves one model (e.g. a node, or a core, or a process) and appears again in some other model. The requirement for the end-to-end analysis is that the "extra-model medium" can give bounded delays for the signal.

V. Proposed Solution

A. Addition of Special Components to RCM

In order to extract the timing model from the distributed real-time applications modeled with RCM, we added special purpose Software Circuits in RCM, i.e., Output Software Circuit (OSWC) and Input Software Circuit (ISWC) [17]. For each message that a node sends to the network there will be one OSWC. Similarly, there will be one ISWC for each message that a node receives from the network. OSWC and ISWC also represent the exit and entry points for the component model respectively.

Moreover, we also introduced a new object in RCM, i.e., the Network Specification (NS) that represents the model of communication in a physical network. NS contains Signal Mapping which includes the following information: How are signals mapped to messages? How many signals a message contains? How are signals encoded in a message at the sender node? How are signals decoded from a message at the receiving node? etc. The model representation of OSWC, ISWC and NS is shown in Figure 4.

The tracing information of all event chains in the modeled system is provided in the Network Specification. In each distributed transaction, the pointers (references) to the input trigger port of OSWC and the output trigger port of ISWC are specified in NS as shown in Figure 4. The grey boxes outside the model, identified as CAN SEND and CAN RECEIVE, are specific for each network protocol. The network protocol considered in this example is CAN.

B. Example: Model of a Node

An example of a node in a distributed real-time system modeled with OSWC and ISWC is shown in Figure 5. The frames that leave the model (sent to CAN SEND)

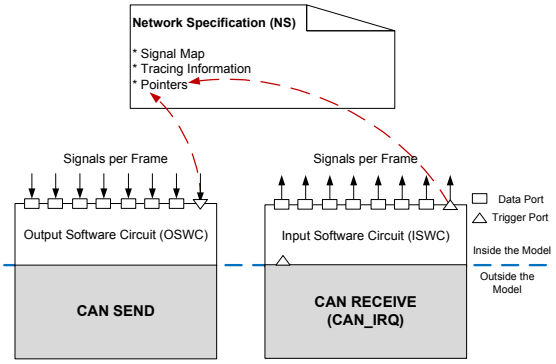

Figure 4. Model representation of OSWC, ISWC and NS

are denoted by S (Send) e.g., $S1$, $S2$ and $S3$. Similarly, all the frames that enter the model (received from CAN RECEIVE) are denoted by R (Receive) e.g., $R1$ and $R2$. All the signals sent in frame $S1$ are provided at the data in-ports of OSWC1. These signals are mapped and encoded into $S1$ by OSWC1 according to the protocol-specific information available in NS. Once the frame is ready, it leaves the model as it is sent to the grey box CAN SEND. In this example, this grey box represents a CAN controller in the node which is responsible for the physical transmission of this frame on the CAN network.

When a frame arrives at the receiving node, it is transferred by the physical network drivers to a grey box (CAN RECEIVE in this example) that produces an interrupt. The frame enters the model and is transferred to the destination ISWC (the tracing information is provided in NS). ISWC extracts the signals from the frame, decodes the data from the frame and encodes it to the RCM data-type. The data is placed on the data out-port of ISWC which is connected to the data in-port of the destination SWC and the corresponding trigger out-port is triggered (the tracing information is provided in NS).

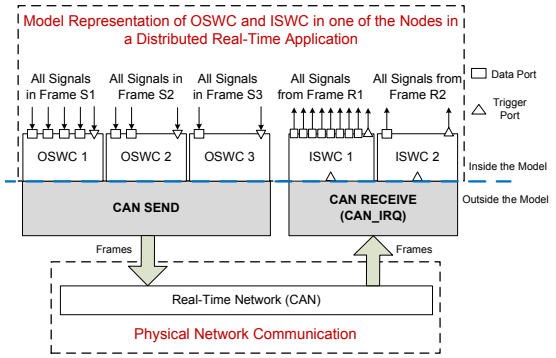

Figure 5. Example node in a distributed real-time system modeled with OSWC and ISWC components

VI. Summary

In this paper, we discussed the problem of tracing event chains while extracting an end-to-end timing model from component-based distributed real-time systems developed with an existing industrial component model, the Rubus Component Model (RCM). The reason for tracing event chains is to extract complete timing information of distributed transactions (in the modeled application) into an end-to-end timing model. The purpose of extracting an end-to-end timing model is to perform holistic response-time analysis which is an important requirement during the development of distributed real-time systems.

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

Currently, we are implementing the analysis framework for holistic response-time analysis in RCM. In future, we plan to validate the solution methodology by making an industrial case study.

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] 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

[2] 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.

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

[4] 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)*, October 2008, pp. 310–317.

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

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

[7] 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.

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

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

[10] 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.

[11] 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.

[12] 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.

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

[14] 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.

[15] 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.

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

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