Implementation of Holistic Response-Time Analysis in Rubus-ICE: Preliminary Findings, Issues and Experiences

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*—There are several issues faced by a developer when holistic response-time analysis (HRTA) is implemented and integrated with a tool chain. The developer has to not only implement the analysis, but also extract unambiguous timing and tracing information from design model. We present an implementation of HRTA as a plug-in for an industrial tool suite Rubus-ICE that is used for component-based development of distributed real-time embedded systems. We present our preliminary findings about implementation issues and highlight our experiences. Moreover, we discuss our plan for testing and evaluating the integration of HRTA as a plug-in in Rubus-ICE.**

***Keywords*-Holistic response-time analysis; distributed real-time embedded systems; component model;**

I. Introduction

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. Holistic Response-Time Analysis (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.

A tool chain for industrial development of component-based distributed real-time embedded (DRE) systems consists of a number of tools such as designer, compiler, builder, debugger, inspector, analyzer, coder, simulator, synthesizer, etc. Often, a tool chain may comprise of tools that are developed by different tool vendors. The implementation of state of the art analysis techniques, e.g., RTA, HRTA, etc. in such a tool chain is not trivial.

In this paper, we discuss the implementation of HRTA as a standalone plug-in in an industrial tool suite Rubus-ICE (Integrated Component development Environment) [1]. We investigate how to practically extract unambiguous timing information from the component model required to carry out HRTA. We present our preliminary findings about implementation issues. We also discuss our plan for testing and evaluating the integration of the HRTA plug-in in Rubus-ICE.

II. Background and Related Work

A. The Rubus Concept

Rubus is a collection of methods and tools for model-based development of dependable embedded real-time systems. The Rubus concept is based around the Rubus Component Model (RCM) [2] and its development environment Rubus-ICE, which includes modeling tools, code generators, analysis tools and run-time infrastructure. The overall goal of Rubus is to be aggressively resource efficient and to provide means for developing predictable and analyzable control functions in resource-constrained embedded systems.

RCM expresses the infrastructure for software functions, i.e., the interaction between 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 is simply: upon triggering, read data on data in-ports; execute the function; write data on data out-ports; and activate the output trigger.

Fig. 1 depicts the sequence of main steps followed in Rubus-ICE from modeling of an application to the generation of code. The component-based design of an application is modeled in the designer tool. Then the compiler compiles the design model into an Intermediate Compiled Component Model (ICCM) file. Then the builder tool runs a set of plug-ins sequentially. Finally, a coder tool generates the code.

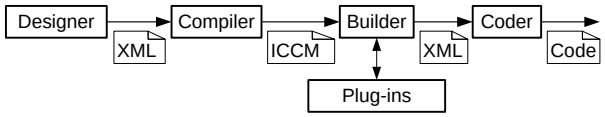

Figure 1. Sequence of steps from design to code generation in Rubus-ICE

B. Plug-in Framework in Rubus-ICE

The plug-in framework in Rubus-ICE [3] facilitates the implementation of state of the art research results in an isolation and their integration as add-on plug-ins with the integrated development environment. A plug-in is interfaced with the builder tool as shown in Fig. 1. The plug-ins are executed sequentially which means that the next plug-in can execute only when the previous plug-in has run to

completion. Hence, each plug-in reads required attributes as an input, runs to completion and finally writes the results to ICCM file. The required and provided services by a plug-in are defined by means of an Application Programming Interface (API). Each plug-in should specify the supported system model, required inputs, provided outputs, error handling mechanisms and a user interface. Fig. 2 shows a conceptual organization of a Rubus-ICE plug-in.

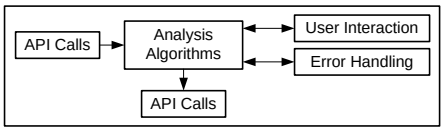

Figure 2. Conceptual organization of a plug-in in Rubus-ICE

C. Holistic Response Time-Analysis (HRTA)

Liu and Layland [4] provided theoretical foundation for analysis of fixed-priority scheduled systems. Joseph and Pandya published the first RTA [5] for the simple task model in [4]. Subsequently, RTA has been applied and extended in a number of ways by the research community. Tindell [6] developed the schedulability analysis for tasks with offsets and it was further extended by Palencia and Gonzalez Harbour [7]. 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. There are many real-time network protocols used in DRE systems. In this paper, we will focus on Controller Area Network (CAN) and its high-level protocols. Tindell et al. [8] developed the schedulability analysis of CAN which has served as a basis for many research projects. Later on, this analysis was revisited and revised by Davis et al. [9].

HRTA combines the analysis of nodes (uniprocessors) and a network. Hence, it computes the response times of event chains that are distributed over several nodes in a DRE system. In this paper, we consider the timing model that corresponds to the holistic schedulability analysis for distributed hard real-time systems [10]. An example distributed transaction in a DRE system is shown in Fig. 3. 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 Task4 (corresponding to the production of a signal for brake actuation).

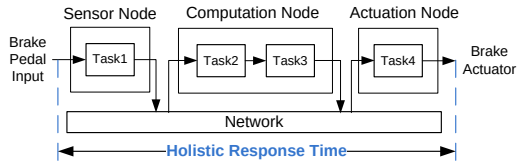

Figure 3. Holistic response-time in a distributed real-time system

III. Implementation

In this section, we discuss the implemented analysis, implementation issues and experiences while implementing and integrating HRTA as a standalone plug-in in Rubus-ICE.

A. Implementation of HRTA in Rubus-ICE

1) Node Analysis: In order to analyze nodes, we implement RTA of tasks with offsets [6], [7] in Rubus-ICE.

2) Network Analysis: As a first step, we have implemented RTA of two different profiles for CAN. Only one of these profiles can be used at a time for the analysis of a DRE application.

1) RTA of CAN [8], [9].
2) RTA of CAN for mixed messages [11].

The next step will be to implement two more analysis profiles for CAN, i.e., RTA of CAN with FIFO queues [12] and RTA of CAN with FIFO Queues for Mixed Messages [13].

B. Implementation Issues and Experiences

1) Extraction of Unambiguous Timing Attributes: One common assumption in HRTA is that the timing attributes required by the analysis are available as an input. However, when HRTA is implemented in a tool chain for the analysis of component-based DRE systems, the implementer has to not only implement the analysis, but also extract unambiguous timing information from the component model and map it to the inputs for the analysis model. Often, the design model contains redundant timing information and hence, it is not trivial to extract unambiguous timing information for HRTA. Examples of timing attributes to be extracted from the design model are worst-case execution times (WCETs), periods, minimum update times, offsets, priorities, deadlines, blocking times, precedence relations, jitters, etc. In [14], we identify all timing attributes of nodes, networks, transactions, tasks and messages that are required by HRTA.

2) Extraction of Tracing Information of Distributed Transactions: In order to perform HRTA, correct tracing information of distributed transactions should be extracted from the design model. For this, we need to have a mapping among signals, data ports and messages. Consider an example of a two-node DRE system modeled with RCM as shown in Fig. 4. Consider the following distributed transaction:
$SWC1 \rightarrow OSWC\_A \rightarrow ISWC\_B \rightarrow SWC2 \rightarrow SWC3$

In this example, our focus is on the network interface components, i.e., Output Software Circuit (OSWC) and Input Software Circuit (ISWC) [15]. In order to compute holistic response time of this distributed transaction, we need to extract unambiguous timing and tracing information from the component model. We identified the need for the following mappings in the component model.

- At the sender node, mapping between signals and input data ports of OSWC components.
- At the sender node, mapping between signals and a message that is sent to the network.
- At the receiver node, mapping between data output ports of ISWC components and the signals to be sent to the desired components.

- At the receiver node, mapping between message received from the network and the signals to be sent to the desired component.
- Mapping between multiple signals and a complex data port. For example, mapping of multiple signals extracted from a received message to a data port that sends a complex signal (structure of signals).
- Mapping of all trigger ports of network interface components along a distributed transaction as shown by a bidirectional arrow in Fig. 4.

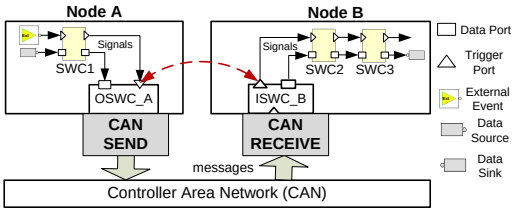

Figure 4. Two-node DRE system modeled in RCM

3) Sequential Execution of Plug-ins in Rubus-ICE: The Rubus plug-in framework allows only sequential execution of plug-ins. This means that a plug-in executes to completion and terminates before the next plug-in is executed. It should be noted that there exists a plug-in in Rubus-ICE that performs RTA of tasks in a node and it is already being used in the industry. There are two options to develop HRTA plug-in for Rubus-ICE, i.e., option A and B as shown in Fig. 5. Option A involves reusing the existing Node RTA plug-in and developing two more plug-ins, i.e., one implementing network RTA algorithms and the other implementing holistic RTA algorithms. In this case HRTA plug-in is very light weight. It iteratively uses the analysis results produced by the node and the network RTA plug-ins and accordingly provides new inputs to them until converging holistic response times are obtained. Option B requires the development of HRTA plug-in from scratch, i.e, implementing the algorithms of node, network and holistic RTA. This option does not provide any reuse of existing plug-ins.

Since, option A allows the reuse of a pre-tested node RTA plug-in (having most complex algorithms compared to network and holistic RTA), it is easy to implement and requires less time to test compared to option B. However, the implementation method in option A is not supported by the plug-in framework of Rubus-ICE because the plug-ins can only be sequentially executed and one plug-in can not execute the other. Hence, we had to select option B for the implementation of HRTA.

4) Impact of Design Decisions in Component Model on the Analysis: Design decisions made in the component model can have indirect impact on the response times computed by the analysis. For example, design decisions could have impact on WCETs and blocking times which in turn have impact on the response times. In order to implement, integrate and test HRTA, the developer needs to understand the design model (component model), analysis model and run-time translation of the design model. In the design model, the architecture of an application is described in terms of software components and their interactions. Whereas in the analysis model, the application is defined in terms of tasks, transactions, messages and timing parameters. At run-time, a task may correspond to a single component or chain of components. The run-time translation of a software component may differ among different component models.

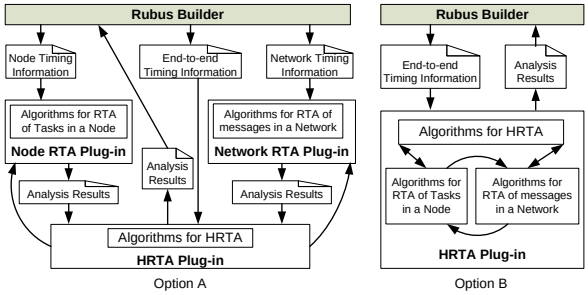

Figure 5. Options to develop HRTA Plug-in for Rubus-ICE

In order to get less pessimistic response times, decisions in component model have to be made to translate the network interface components (OSWC and ISWC) either as separate tasks or as a part of the task corresponding to the software component immediately connected to them. Another issue is that if the developer and integrator of HRTA plug-in are two different people with different backgrounds, e.g., research and industrial, the integration testing and verification becomes a difficult and time consuming activity.

5) Analysis of DRE Application with Multiple Networks: In a DRE application, a node may be connected to more than one networks. If a transaction is distributed over more than one networks, the holistic response time of the transaction involves the analysis of all networks in the system. Consider an example of a DRE system with two networks, i.e., CAN and LIN as shown in Fig. 6. There are five nodes in the system. Node 3 is a gateway node that is connected to both the networks. Consider a transaction in which $task1$ in $Node1$ sends a message to $task1$ in $Node5$ via $Node3$. Computation of holistic response time of this transaction will involve the computation of message response times in both CAN and LIN networks.

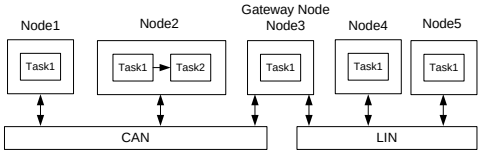

Figure 6. Multiple networks in a DRE system

As a first implementation step, we assume that all nodes in a DRE system are connected to a single network. If an application contains more than one network, we will divide it into sub-applications (with each having a single network) and analyze them separately.

6) Feedback from HRTA Plug-in: We identified that it is very important to provide the progress report of HRTA

plug-in to user during the analysis. The algorithms in HRTA iteratively run the algorithms of node RTA and network RTA until converging values of the response times are computed or the computed response times exceed the deadlines. It is important to display the number of iterations performed. A user should be provided the control to stop the plug-in at any time. If the analysis results indicate that the system is unschedulable, it will be interesting to provide suggestions to the user to make the system schedulable.

7) Requirement for Continuous Collaboration between Integrator and Developer: Our experience of integrating HRTA plug-in with Rubus-ICE shows that there is a need of continuous collaboration between the developer of the plug-in and its integrator especially in the phase of integration testing (see next section).

IV. Test Plan

In this section we discuss our test plan for both standalone and integration testing of HRTA plug-in. Error handling and sanity checking routines make a significant part of the implementation. The purpose of these routines is to detect and isolate faults and present them to the user during the analysis. Our test plan contains following set of error handling routines.

- Testing of all inputs: attributes of all nodes, transactions, tasks, networks and messages in the system.
- Testing of linking and tracing information of all distributed transactions in the system.
- Testing of intermediate results that are iteratively used as inputs (e.g., a message inheriting the worst-case response time of the sender tasks as a release jitter).
- Testing of overload conditions during the analysis (e.g., processor utilization exceeding 100%).
- Testing of variable overflow during the analysis.

A. Standalone Testing

Standalone testing means testing of HRTA implementation before it is integrated with the Rubus builder tool as a plug-in. In other words, it refers to the testing of HRTA in isolation. We have already finished standalone testing. We used following input methods for standalone testing.

1) Hard coded input test vectors.
2) Test vectors are read from external files.
3) Test vector generator (a separate program).

B. Integration Testing

Integration testing means testing of HRTA implementation after integrating it with the Rubus builder tool as a plug-in. Although standalone testing is already performed, the integration of HRTA with the Rubus-ICE may induce unexpected errors. Currently we are doing integration testing. Our experience shows that integration testing is much more difficult and time consuming compared to standalone testing. We will use following input methods for integration testing.

1) Test vectors are read from external files.
2) Test vectors are manually written in ICCM file (see Fig. 1) to make it appear as if test vectors were extracted from the modeled application.
3) Inputs should be automatically extracted from a DRE application modeled with Rubus component model.

V. Summary

We presented an implementation of state of the art holistic response-time analysis as a plug-in for the industrial tool suite Rubus-ICE. We discussed our preliminary findings regarding implementation issues. We discussed our experiences and presented our test plan. We have completed the implementation and performed standalone testing. Currently we are doing integration testing of the plug-in in Rubus-ICE. In future, we plan to model and analyze a large industrial DRE application complemented with benchmarks.

Acknowledgement

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

References

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

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

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

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

[5] 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.

[6] 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.

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

[8] 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.

[9] 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.

[10] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocess. Microprogram.*, vol. 40, pp. 117–134, April 1994.

[11] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "Extending schedulability analysis of controller area network (CAN) for mixed (periodic/sporadic) messages," in *Emerging Technologies Factory Automation (ETFA), 2011 IEEE 16th Conference on*, sept. 2011, pp. 1 –10.

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

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

[14] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "Extraction of end-to-end timing model from component-based distributed real-time embedded systems," in *Time Analysis and Model-Based Design, from Functional Models to Distributed Deployments (TiMoBD) workshop located at Embedded Systems Week*. Springer, October 2011, p. 6.

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