Support for End-to-End Response-Time and Delay Analysis in the Industrial Tool Suite: Issues, Experiences and a Case Study

Saad Mubeen[1], Jukka Mäki-Turja[1,2], and Mikael Sjödin[1]

[1] Mälardalen Real-Time Research Centre (MRTC), Mälardalen University, Sweden
[2] Arcticus Systems, Järfälla, Sweden
{saad.mubeen, jukka.maki-turja, mikael.sjodin}@mdh.se

Abstract. In this paper we discuss the implementation of the state-of-the-art end-to-end response-time and delay analysis as two individual plug-ins for the existing industrial tool suite Rubus-ICE. The tool suite is used for the development of software for vehicular embedded systems by several international companies. We describe and solve the problems encountered and highlight the experiences gained during the process of implementation, integration and evaluation of the analysis plug-ins. Finally, we provide a proof of concept by modeling the automotive-application case study with the existing industrial model (the Rubus Component Model), and analyzing it with the implemented analysis plug-ins.

Keywords: real-time systems, response-time analysis, end-to-end timing analysis, component-based development, distributed embedded systems.

1. Introduction

Often, an embedded system needs to interact and communicate with its environment in a timely manner, i.e., the embedded system is a real-time system. For such a system, the desired and correct output is one which is logically correct as well as delivered within a specified time. Many real-time systems are also safety critical which means that the system failure can result in catastrophic consequences such as endangering human life or the environment. The safety-critical nature of these systems requires evidence that the actions by them will be provided in a timely manner, i.e., each action will be taken at a time that is appropriate to the environment of the system. Therefore, it is important to make accurate predictions of the timing behavior of these systems.

In order to provide evidence that each action in the system will meet its deadline, *a priori* analysis techniques such as schedulability analysis have been developed by the research community. Response Time Analysis (RTA) [17, 47] is one of the methods to check the schedulability of a system. It calculates upper bounds on the response times of tasks or messages in a real-time system or a network respectively. Holistic Response-Time Analysis (HRTA) [50, 49, 43] is an academic well established schedulability analysis technique to calculate

upper bounds on the response times of task chains that may be distributed over several nodes in a Distributed Real-time Embedded (DRE) system.

A task chain is a sequence of more than one task in which every task (other than the first) receives a trigger, data or both from its predecessor. One way to classify these chains is as trigger chains and data chains. In trigger chains, there is only one triggering source (e.g, event, clock or interrupt) that activates the first task in the chain. The rest of the tasks are activated by their predecessors. In data chains, tasks are activated independent of each other, often with distinct periods. Each task (except the first) in these chains receives data from its predecessor. The first task in a data chain may receive data from the peripheral devices and interfaces, e.g., signals from the sensors or messages from the network interfaces. The end-to-end timing requirements on trigger chains are different from those on data chains. If a system is modeled with trigger chains only, it is called a single-rate system. On the other hand, if the system contains at least one data chain with different clocks then the system is said to be multi-rate.

In order to predict complete timing behavior of multi-rate real-time systems [21], the end-to-end delays should also be computed along with the holistic response times. For this purpose, the research community has developed the End-to-End Delay Analysis (E2EDA). In [21], the authors have a view that almost all automotive embedded systems are multi-rate systems. The industrial tools used for the development of such systems should be equipped with the state-of-the-art timing analysis.

The process of transferring such academic research results to the tools for industrial use can be challenging. A tool chain for the industrial development of component-based DRE systems consists of a number of tools such as designer, compiler, builder, debugger, simulator, etc. Often, a tool chain may comprise of tools that are developed by different tool vendors. The implementation of state-of-the-art complex real-time analysis techniques such as RTA, HRTA and E2EDA in such a tool chain is non-trivial because there are several challenges that are encountered apart from merely coding and testing the analysis algorithms. These challenges and corresponding solutions that we propose are central to this paper.

1.1. Goals and Paper Contributions

In this paper, we discuss the implementation of holistic response time analysis and end-to-end [3] delay analysis as two plug-ins in the existing industrial tool suite Rubus-ICE (Integrated Component development Environment) [1]. Our goals in this paper are as follows.

---

[3] The terms "holistic" and "end-to-end" mean the same thing. In order to be consistent with the previous work and naming conventions used in the existing industrial tools, we will use "holistic" with response-times and "end-to-end" with delays.

1. Transfer the state-of-the-art real-time analysis results, i.e., holistic response-time analysis and end-to-end delay analysis to the existing tools for the industrial use.

2. Discuss and solve several problems encountered during the implementation, integration and evaluation of HRTA and E2EDA as two individual plug-ins for Rubus-ICE.

3. Discuss the experiences gained during the implementation, integration and evaluation of the HRTA and E2EDA plug-ins.

4. Provide a proof of concept by conducting an automotive-application case study.

1.2. Paper Layout

The rest of the paper is organized as follows. Section 2 presents the background and related work. Section 3 discusses the end-to-end timing requirements and the analysis that we implemented in Rubus-ICE. Section 4 describes the challenges encountered, solutions proposed and experiences gained during the implementation and integration of the HRTA and E2EDA plug-ins. Section 5 presents our test plan. In Section 6, we present a case study by modeling and analyzing the automotive DRE application. Section 7 concludes the paper and presents the future work.

2. Background and Related Work

2.1. Relation to Authors' Previous Work

This work is the extension of our previous work [38] in which we discussed the implementation of only HRTA plug-in for the Rubus-ICE. In this paper, we implement E2EDA as a second plug-in. As compared to our previous work, this paper presents a detailed discussion on the end-to-end timing requirements in the industrial DRE systems. We also discuss the algorithm of end-to-end delay analysis and its conceptual organization in Rubus-ICE. Further, we discuss several challenging problems that were encountered during the implementation, integration and evaluation of the E2EDA plug-in. Moreover, we discuss the proposed solutions and gained experiences during the process of transferring state-of-the-art research results to the industrial tool suite. For the sake of completeness, we also revisit the problems and their solutions corresponding to the HRTA plug-in.

We also reconducted the case study. This is because the automotive DRE application (Autonomous Cruise Control system) considered in the previous work was modeled with only trigger chains. This limited the usability of our modeling and analysis tools because many automotive embedded systems in the industry are build using data and mixed chains as well. Therefore, we remodeled the same automotive-application with trigger, data and mixed chains.

Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin

We also analyzed it with both the HRTA and E2EDA plug-ins. With the addition of E2EDA plug-in, a complete end-to-end timing analysis of DRE systems can be performed. Thus, the scope and usability of Rubus tools has widened with the addition of HRTA and E2EDA plug-ins.

2.2. 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 [1] 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 [2, 13, 7, 5]. The Rubus concept is based around the Rubus Component Model (RCM) [25] 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 software functions in terms of data and control flow separately. The control flow is expressed by triggering objects such as internal periodic clocks, interrupts, internal and external events. In RCM, the basic component is called Software Circuit (SWC). The execution semantics of the SWC 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.

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. Recently, we extended RCM for the development of DRE systems by introducing new components [31, 40, 34]. A detailed comparison of RCM with several component models is presented in [40].

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 Rubus Designer tool. Then the compiler compiles the design model into the Intermediate Compiled Component Model (ICCM). After that the builder tool sequentially runs a set of plug-ins. Finally, a coder tool generates the code.
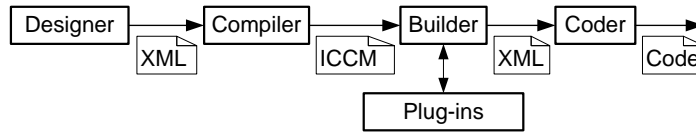
Fig. 1. Sequence of steps from design to code generation in Rubus-ICE

2.3. 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 response time analysis and shared stack analysis.

2.4. Plug-in Framework in Rubus-ICE

The plug-in framework in Rubus-ICE [28] facilitates the implementation of state-of-the-art research results in isolation (without needing Rubus tools) and their integration as add-on plug-ins (binaries or source code) 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 inputs, runs to completion and finally writes the results to the ICCM file. The Application Programming Interface (API) defines the services required and provided by a plug-in. Each plug-in specifies the supported system model, required inputs, provided outputs, error handling mechanisms and a user interface. Fig. 2 shows a conceptual organization of a Rubus-ICE plug-in.
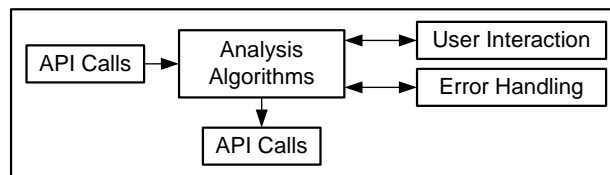

Fig. 2. Conceptual organization of a plug-in in Rubus-ICE

2.5. Response-Time Analysis

RTA of Tasks in a Node. Liu and Layland [29] provided theoretical foundation for analysis of fixed-priority scheduled systems. Joseph and Pandya published the first RTA [27] for the simple task model presented in [29]. Subsequently, it has been applied and extended in a number of ways by the research community. 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 [41]. In [41], 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.

Tindell [49] developed the schedulability analysis for tasks with offsets for fixed-priority systems. It was extended by Palencia and Gonzalez Harbour [43]. Later, Mäki-Turja and Nolin [30] reduced pessimism from RTA developed in [49, 43] and presented a tighter RTA for tasks with offsets by accurately modeling inter-task interference. In [21, 48], the authors point out that the existing RTA does not target general multi-rate systems. We implemented tighter version of RTA of tasks with offsets [30] as part of the end-to-end response-time and delay analysis.

RTA of Messages in a Network. There are many protocols such as Controller Area Network (CAN), Time-Triggered CAN (TTCAN), FlexRay, etc., that are used for network communication in DRE systems. To stay focussed on the automotive or vehicular domain, we will consider only CAN and its high-level protocols. Tindell et al. [51] 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. [19].

The analysis in [51, 19] assumes that all CAN device drivers implement priority-based queues. In [20] Davis et al. pointed out that this assumption may become invalid when some nodes in a CAN network implement FIFO queues. Hence, they extended the analysis of CAN with FIFO queues as well. In this work, the message deadlines are assumed to be smaller than or equal to the corresponding periods. In [18], Davis et al. lifted this assumption by supporting the analysis of CAN messages with arbitrary deadlines. Furthermore, they extended their work to support RTA of CAN for FIFO and work-conserving queues.

However, the existing analysis does not support mixed messages which are implemented by several high-level protocols for CAN. In [33, 37, 32], Mubeen et al. extended the existing analysis to support RTA of mixed messages in the CAN network where some nodes use FIFO queues while others use priority queues. Later on, Mubeen et al. [39] extended the existing analysis for CAN to support mixed messages that are scheduled with offsets in the controllers that implement priority-ordered queues. In this work we will consider all of the above analyses as part of the end-to-end response-time and delay analysis.

Holistic RTA. It combines the analysis of nodes (uniprocessors) and networks. In this paper, we consider the end-to-end timing model that corresponds to the holistic schedulability analysis for DRE systems [50]. In [44], Pop et al. provide a holistic schedulability analysis of distributed embedded systems in which tasks are both time- and event-triggered. The analysis is developed for ST/DYN bus protocol that uses static and dynamic phases for sending messages. As compared to this approach, we implement the holistic analysis of [50] because it provides the flexibility to use several network-communication protocols used in the automotive domain. In [35], we discussed our preliminary findings about implementation issues that are encountered when HRTA is transferred to the industrial tools.

End-to-end Delay Analysis. Stappert et al. [48] formally described end-to-end timing constrains for multi-rate systems in the automotive domain. In [21], Feiertag et al. presented a framework (developed in TIMMO project [16]) for the computation of end-to-end delays for multi-rate automotive embedded systems. Furthermore, they emphasized on the importance of two end-to-end latency semantics, i.e., "maximum age of data" and "first reaction" in control systems and body electronics domains respectively. A scalable technique, based on model checking, for the computation of end-to-end latencies is described in [45]. In this work, we will implement the end-to-end delay analysis [21] as a plug-in for the Rubus-ICE tool suite.

2.6.   Tools for End-to-end Timing Analysis of DRE Systems

We briefly discuss few tool suites that provide similar real-time analysis support for DRE systems. The MAST tool suite [6] implements a number of state-of-the-art analysis algorithms for DRE systems. Among them is the offset-based analysis algorithm [49, 43] whose tighter version [30] is implemented as part of the end-to-end response-time and delays analysis in Rubus-ICE. The MAST model also allows visual modeling and analysis of real-time systems in a Unified Modeling Language (UML) design environment.

The Volcano Family [10] is a bunch of tools for designing, analyzing, testing and validating automotive embedded software systems. Among them, Volcano Network Architect (VNA) [12] is a communication design tool that supports the analysis of Local Interconnect Network (LIN) and CAN networks. It also supports end-to-end timing analysis of a system with more than one network. It implements RTA of CAN developed by Tindell et al. [51].

SymTA/S [24] is a tool for model-based timing analysis and optimization. It implements several real-time analysis techniques for single-node, multiprocessor and distributed systems. It supports RTA of software functions, RTA of bus messages and end-to-end timing analysis of both single-rate and multi-rate systems. It is also integrated with the UML development environment to provide a timing analysis support for the applications modeled with UML [22].

Vector [11] is a tools provider for the development of networked electronic systems in the automotive and related domains. In the Vector tool family, CANoe

[3] is a tool for the development, testing and analysis of ECU (Electronic Control Units) networks and individual ECUs. It supports various protocols for network communication including CAN, LIN, MOST, Flexray, Ethernet and J1708. Network Designer CAN is another tool by Vector that is used to design the architecture and perform timing analysis of CAN network.

RAPID RMA [8] implements several scheduling schemes and supports end-to-end analysis for single- and multiple-node real-time systems. It also allows real-time analysis support for the systems modeled with Real-Time CORBA [46].

The Rubus-ICE tool suite allows a developer to specify timing information and perform end-to-end response time and delay analysis at the modeling phase during component-based development of DRE systems. To the best of our knowledge, Rubus-ICE is the first and only tool suite that implements RTA of mixed messages in CAN [33], RTA of mixed messages with offsets [39] and a tighter version of offset-based RTA algorithm [30] as part of the end-to-end response time and delay analysis.

3. End-to-end Timing Requirements and Implemented Analysis in Rubus-ICE

3.1. End-to-end timing requirements in trigger chains

A real-time system can be modeled with trigger chains (see Fig.4 and Fig.5), data chains (see Fig.6 and Fig.8) or a combination of both. The end-to-end timing requirements on trigger chains are different from those on data chains. If the system is modeled with trigger chains then the interest, from the schedulability point of view, lies in the calculation of end-to-end or holistic response times and their comparison with corresponding deadlines. Hence, end-to-end deadline requirements placed on trigger chains correspond to their holistic response times. If holistic response times of all trigger chains are less than or equal to corresponding deadlines, the system is considered schedulable.

The holistic response-time analysis calculates the response times of event chains that are distributed over several nodes (also called distributed transactions) in a DRE system. An example of a 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 in the sensor node) and the response time of Task4 (corresponding to the production of brake actuation signal in the actuation node).

Examples. An example of a trigger chain that consists of three components is shown in Fig. 4. Assume that each component corresponds to a task at run-time. When task τ_{SWC_A} finishes its execution, it triggers τ_{SWC_B}. Similarly, τ_{SWC_C} can only be triggered by τ_{SWC_B} after finishing its execution. There cannot be multiple outputs corresponding to a single input signal. In fact, there will always be one output of the chain corresponding to the input trigger. Hence,
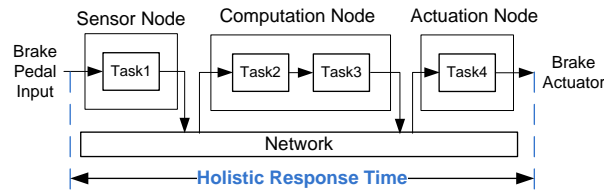
Fig. 3. Holistic response-time in a distributed real-time system

the end-to-end timing requirements correspond to the holistic response times. In order to provide a comparison of holistic response time in a trigger chain with the end-to-end delays in a data chain, assume that the trigger chain shown in Fig. 4 is the only chain of tasks in the system. Let the priorities of all tasks be the same while WCET of each task is $1ms$. The holistic response time of this trigger chain is equal to the response time of τ_{SWC_C} which is, intuitively, equal to $3ms$.
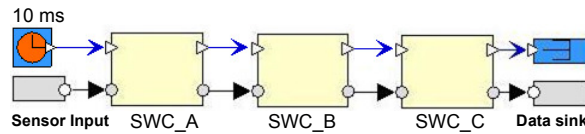
Fig. 4. RCM model of a trigger chain in a single-node real-time system

Distributed real-time systems can also be modeled with trigger chains. Consider a model of a two-node distributed real-time system modeled with RCM as shown in Fig. 5. There is only one triggering ancestor in node A that activates SWC_A which, in turn, triggers $OSWC\_A$ component that is responsible for sending a message to CAN. The $ISWC\_C$ component in only activated when an interrupt is raised due to the arrival of a CAN message at node C. Hence, these three components form a distributed trigger chain. Once again, the end-to-end timing requirements correspond to the holistic response times.

3.2. End-to-end timing requirements in data chains

As compared to the systems which are modeled with trigger chains, merely computing the holistic response times and comparing them with the end-to-end deadlines is not sufficient to predict the complete timing behavior of multi-rate real-time systems which are modeled with data chains. There may be over- and under-sampling in such systems because the individual tasks are activated by independent clocks, often with different periods. Since data is transferred among tasks and messages within a data chain by means of asynchronous buffers, there exist different semantics of end-to-end delay in a data chain.
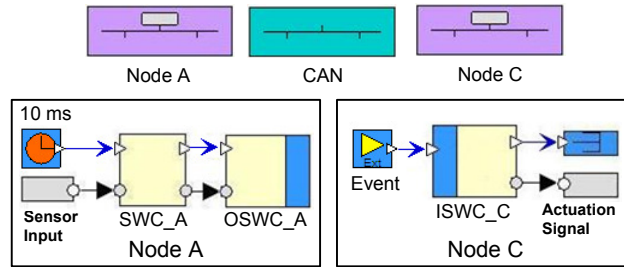
Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin

Fig. 5. RCM model of a distributed trigger chain in a DRE system

These buffers are often of a non-consuming type which means the data stays in the buffer after it is read by the reader task. Moreover, the data in the buffer can be overwritten by the writer task with new values before the previous value was read by the reader task. Therefore, some input values in the data buffers can be overwritten by new values, and hence the effect of the old input values may never propagate to the output of a data chain. Further, it is also possible to have several duplicates of the output of a data chain corresponding to a particular input.

The end-to-end timing requirements in multi-rate real-time systems, especially in the automotive domain, are placed on the first reaction to the input and age of the data received at the output [21]. Hence, it is important to calculate the end-to-end delays in these systems. The end-to-end delay in a data chain refers to the time elapsed between the arrival of a signal at the first task and production of corresponding output signal by the last task in the chain (provided the information corresponding to the input signal has traversed the chain from first to last task) [45].

In a single-rate real-time system that contains only trigger chains, tasks in a chain are not activated by independent events, in fact, there is only one activating event in the chain. Hence, the holistic response times and end-to-end delays will have equal values. On the other hand, these values are not the same in multi-rate real-time systems that are modeled with data chains. Therefore, a complete analysis of a real-time system modeled with data chains requires the calculation of not only holistic response times but also end-to-end delays.

Examples. A multi-rate real-time system modeled with three SWCs in RCM is shown in Fig. 6. These SWCs are activated by independent clocks with different periods, i.e., 8ms, 16ms and 4ms respectively. SWC_A reads the input signals from the sensors while SWC_C produces the output signals for the actuators. Assume that each SWC will be allocated to an individual task by the run-time environment generator. Also assume that WCET of each task is $1\,ms$.

The time line corresponding to the run-time execution of the three tasks (corresponding to three SWCs) is depicted in Fig. 7. It can be seen that there
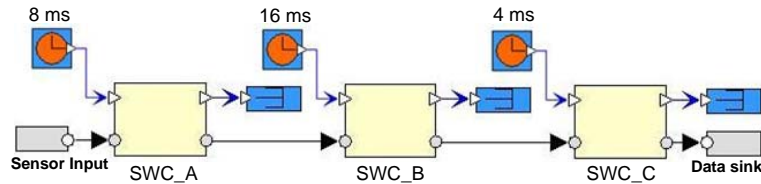
Fig. 6. RCM model of a data chain in a single-node real-time system

are multiple outputs corresponding to a single input signal. The four end-to-end delays are identified in Fig. 7.

Last In First Out (LIFO). This delay is equal to the time elapsed between the current non-overwritten release of task τ_A (input of the data chain) and corresponding first response of task τ_C (output of the data chain).

Last In Last Out (LILO). This delay is equal to the time elapsed between the current non-overwritten release of task τ_A (input of the data chain) and corresponding last response of task τ_C (output of the data chain). This delay is identified as "Data Age"[4] in [21]. Data age specifies the longest time data is allowed to age from production by the initiator until the data is delivered to the terminator. This delay finds its importance in control applications where the interest lies in the freshness of the produced data. For a data chain in a control system that initiates with a sensor input and terminates by producing an actuation signal, it is very important to ensure that the actuator signal does not exceed a maximum age [21].

Generally speaking, we consider the last non-overwritten input that actually propagates through the data chain towards the output in the case of both LIFO and LILO delays.

First In First Out (FIFO). This delay is equal to the time elapsed between the previous non-overwritten release of task τ_A (input of the data chain) and first response of task τ_C (output of the data chain) corresponding to the current non-overwritten release of task τ_A. Assume that a new value of the input is available in the input buffer of task τ_A "just after" the release of the second instance of task τ_A (at time $8ms$). Hence, the second instance of task τ_A "just misses" the read of the new value from its input buffer. This new value has to wait for the next instance of task τ_A to travel towards the output of the data chain. Therefore, the new value will be read by the third and forth instances of task τ_A. The first output corresponding to the new value (arriving just after $8ms$) will appear at the output of the chain at $34ms$. This will result in the FIFO delay of $26ms$ as shown in Fig. 7. This phenomenon is more obvious in the case of distributed embedded systems where a task in the receiving node may just miss to read fresh signals from a message that is received from the network.

---

[4] We will use the term "Data Age delay" to refer to LILO delay throughout the paper.

This delay is identified as "first reaction to data or Data Reaction"[5] in [21]. Data reaction delay is the longest allowed reaction time for data produced by the initiator to be delivered to the terminator. This delay finds its importance in the button-to-reaction applications in body electronics domain where first reaction to input is important.

First In Last Out (FILO). This delay is equal to the time elapsed between the previous non-overwritten release of task τ_A (input of the data chain) and last response of task τ_C (output of the data chain) corresponding to the current non-overwritten release of task τ_A. The reasoning about "just missing" a fresh input that we discussed in the case of FIFO delay is also applicable in the case of FILO delay.
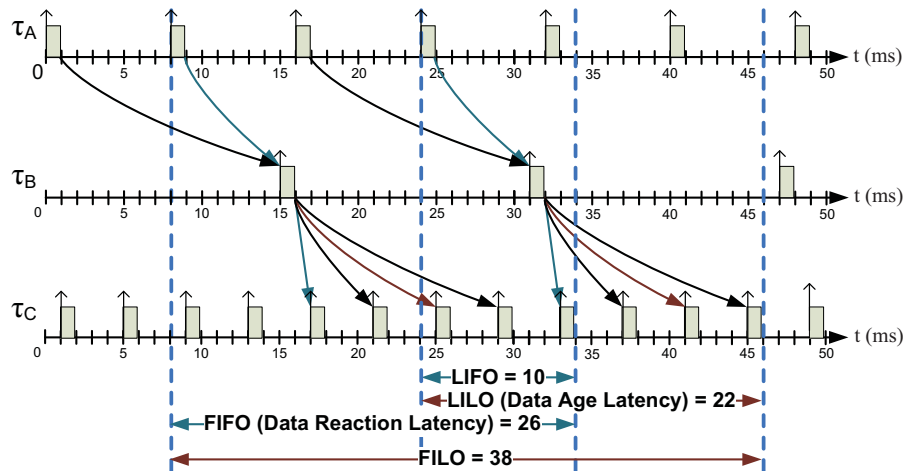
Fig. 7. End-to-end delays for a data chain in a real-time system

In the case of distributed real-time systems, data chains may also be distributed over more than one node. Consider a model of a two-node distributed real-time system modeled with RCM as shown in Fig. 8. The nodes are connected to the CAN network. The internal model of the nodes is also shown in Fig. 8. In Node A, SWC_A is triggered by a clock with a period of 8ms. The $OSWC\_A$ component that is responsible for sending a message to the network is triggered by another clock with a period of 16ms. $ISWC\_C$ is a component that receives a message from the network and is activated by a clock with a period of 4ms. Assume that each component is allocated to a separate task at run-time, i.e., the components SWC_A, $OSWC\_A$ and $ISWC\_C$ are allocated to

---

[5] We will use the term "Data Reaction delay" to refer to FIFO delay throughout the paper.

tasks τ_A, τ_B and τ_C respectively. Since, the system consists of tasks with similar activation patterns and periods as compared to the tasks in the single-node real-time system example discussed above, it can be scheduled in a similar manner as indicated by τ_A, τ_B and τ_C in Fig. 7. The end-to-end delays are also defined in a similar fashion.
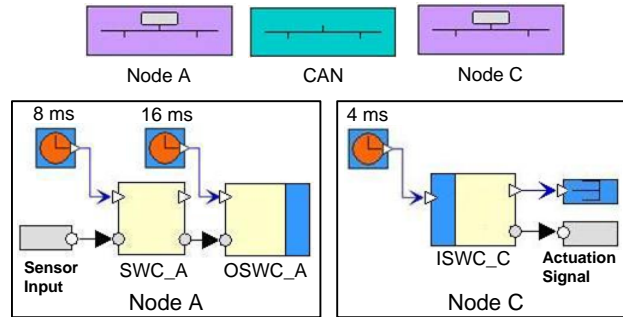

Fig. 8. RCM model of a distributed data chain in a DRE system

3.3. Implemented Holistic Response-Time Analysis

We implemented HRTA as a plug-in in Rubus-ICE. The plug-in can be used to compute the response times of individual tasks in a node, messages in a network and Distributed Transactions (DTs) in a distributed real-time system.

In order to analyze tasks in each node, we implement RTA of tasks with offsets developed by [49, 43] and improved by [30]. We implement the network RTA that supports the analysis of CAN and its high-level protocols. It is based on the following RTA profiles for CAN.

1. RTA of CAN [51, 19].
2. RTA of CAN for mixed messages [33].
3. RTA of CAN for mixed messages with offsets [39][6].

The above analyses assume that CAN nodes implement priority-ordered queues. The next step, as a future work, will be the implementation of CAN analysis that also supports FIFO ordered queues, i.e., RTA of CAN with FIFO and work-conserving queues [18, 20] and RTA of CAN with FIFO Queues for Mixed Messages [37].

The pseudocode of HRTA algorithm is shown in Algorithm 1. The HRTA algorithm iteratively runs the algorithms for node and network analyses. In the first step, release jitter of all messages and tasks in the system is assumed to be

---

[6] The analysis of this profile is implemented as a standalone analyzer whose integration with Rubus-ICE is a work in progress

zero. The response times of all messages in the network and all tasks in each node are computed. In the second step attribute inheritance is carried out. This means that each message inherits a release jitter equal to the difference between the worst- and best-case response times of its sender task (computed in the first step). Similarly, each task that receives the message inherits a release jitter equal to the difference between the worst- and best-case response times of the message (computed in the first step). In the third step, response times of all messages and tasks are computed again. The newly computed response times are compared with the response times previously computed in the first step. The analysis terminates if the values are equal otherwise these steps are repeated. The conceptual view of HRTA that we implemented in Rubus-ICE is shown in Fig. 9.

Algorithm 1 Algorithm for holistic response-time analysis

1: **begin**
2: $RT_{Prev} \leftarrow 0$ ▷ Initialize all Response Times (RTs) to zero
3: $Repeat \leftarrow TRUE$
4: **while** $Repeat = TRUE$ **do**
5:     **for all** Messages_and_tasks_in_the_system **do**
6:         $Jitter_{Msg} \leftarrow (WCRT_{Sender\_task} - BCRT_{Sender\_task})$ ▷ WCRT: Worst-Case
    Response Time, BCRT: Best-Case Response Time
7:         $Jitter_{Receiver\_task} \leftarrow (WCRT_{Msg} - BCRT_{Msg})$
8:         COMPUTE_RT_OF_ALL_MESSAGES()
9:         COMPUTE_RT_OF_ALL_TASKS_IN_EVERY_NODE()
10:        **if** $RT > RT_{Prev}$ **then**
11:            $RT_{Prev} \leftarrow RT$
12:            $Repeat \leftarrow TRUE$
13:        **else**
14:            $Repeat \leftarrow FALSE$
15:        **end if**
16:    **end for**
17: **end while**
18: **end**

3.4. Implemented End-to-end Delay Analysis

We implemented the end-to-end delay analysis that is derived in [21] as the E2EDA plug-in for Rubus-ICE. This analysis implicitly requires the calculation of response times of individual tasks, messages and holistic response times of task chains. For example, the calculation of four end-to-end delays for the multi-rate real-time system shown in Fig. 6 requires the response time of the task τ_C (corresponding to the component SWC_C) and the activation times of tasks τ_A and τ_C. Similarly, the calculation of four end-to-end delays for the multi-rate DRE system shown in Fig. 8 requires the calculation of the response time of
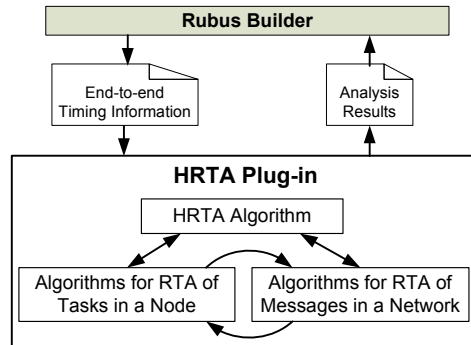
Fig. 9. Conceptual view of the HRTA plug-in in Rubus-ICE

the task τ_C in node C and the activation times of tasks τ_A and τ_C in nodes A and C respectively. Since, the HRTA plug-in is able to calculate response times of tasks, network messages and task chains, we reuse the analysis results computed by the HRTA plug-in as an input to the E2EDA plug-in as shown in Fig. 10. The pseudocode of E2EDA algorithm[7] is shown in Algorithm 2.
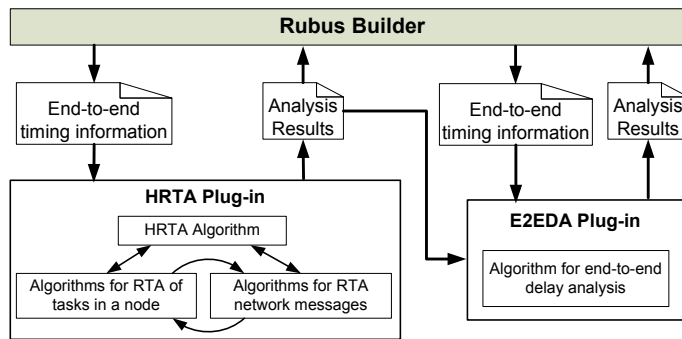

Fig. 10. Conceptual view of the E2EDA plug-in in Rubus-ICE

4. Encountered Problems, Proposed Solutions and Gained Experiences

In this section we discuss several problems encountered during the process of implementation and integration of HRTA and E2EDA as plug-ins for the Rubus-ICE tool suite. We also present our solution to each individual problem. More-

---

[7] [21] should be referred for detailed analysis.

Algorithm 2 Algorithm for end-to-end delay analysis

1: **begin**
2: GET_RT_OF_ALL_TASKS_MESSAGES_TASK_CHAINS() ▷ Get the analysis results from the HRTA plug-in
3: FIND_ALL_VALID_TIMED_PATHS() ▷ Timed Path (TP) is a sequence of task instances from input to output. A TP is valid if information flow among tasks is possible [21], e.g., $[\tau_A(1^{st}\text{instance}), \tau_B(1^{st}\text{instance}), \tau_C(5^{th}\text{instance})]$ in Fig. 7 is a valid TP. On the other hand, TP $[\tau_A(1^{st}\text{instance}), \tau_B(1^{st}\text{instance}), \tau_C(1^{st}\text{instance})]$ in Fig. 7 is invalid because information cannot flow between $\tau_B(1^{st}\text{instance})$ and $\tau_C(1^{st}\text{instance})$
4: **procedure** COMPUTE_FF_DELAY(FF_TP)
5:     FF_delay = α_n(instance) + δ_n(instance) - α_1(instance) ▷ α_n(instance): Activation time of the corresponding instance of the nth task in timed path FF_TP
        ▷ δ_n(instance): Response time of the corresponding instance of the nth task in timed path FF_TP
6:     **return** FF_delay
7: **end procedure**
        ▷ The above mentioned procedure calculates FF_{Delay} only. [21] should be referred for the calculation of the rest of the delays
8: **for all** Delay_constraints_specified_in_the_system **do**
9:     $FF_{Delay} \leftarrow 0, FL_{Delay} \leftarrow 0, LF_{Delay} \leftarrow 0, LL_{Delay} \leftarrow 0$ ▷ Initialize all delays
10:     COMPUTE_ALL_REACHABLE_TIMED_PATHS()   ▷ All those paths from input to output in which the changes in input actually travel towards the output, e.g., $[\tau_A(2^{nd}\text{instance}), \tau_B(1^{st}\text{instance}), \tau_C(5^{th}\text{instance})]$ in Fig. 7
11:     $FF\_TP_{count} \leftarrow$ GET_ALL_FF_TPS()   ▷ TP: Timed Path, FF: First to First
12:     $FL\_TP_{count} \leftarrow$ GET_ALL_FL_TPS()   ▷ FL: First to Last
13:     $LF\_TP_{count} \leftarrow$ GET_ALL_LF_TPS()   ▷ LF: Last to First
14:     $LL\_TP_{count} \leftarrow$ GET_ALL_LL_TPS()   ▷ LL: Last to Last
15:     **for** i:=1 **do** FF_TP_{count}
16:         **if** COMPUTE_FF_DELAY(i) $> FF_{Delay}$ **then**
17:             $FF_{Delay} \leftarrow$ COMPUTE_FF_DELAY()
18:         **end if**
19:     **end for**
20:     **for** i:=1 **do** FL_TP_{count}
21:         **if** COMPUTE_FL_DELAY(i) $> FL_{Delay}$ **then**
22:             $FL_{Delay} \leftarrow$ COMPUTE_FL_DELAY()
23:         **end if**
24:     **end for**
25:     **for** i:=1 **do** LF_TP_{count}
26:         **if** COMPUTE_LF_DELAY(i) $> LF_{Delay}$ **then**
27:             $LF_{Delay} \leftarrow$ COMPUTE_LF_DELAY()
28:         **end if**
29:     **end for**
30:     **for** i:=1 **do** LL_TP_{count}
31:         **if** COMPUTE_LL_DELAY(i) $> LL_{Delay}$ **then**
32:             $LL_{Delay} \leftarrow$ COMPUTE_LL_DELAY()
33:         **end if**
34:     **end for**
35: **end for**
36: **end**

---

over, we discuss the summary of the experiences that we gained while transferring theoretical research results (i.e., HRTA and E2EDA) to the industrial tool suite.

4.1. Extraction of Unambiguous Timing Information

One common assumption in end-to-end response time and delay analyses is that the timing attributes required by the analysis are available as input. However, when these analyses are implemented in a tool chain used for the component-based development of DRE systems, the implementer has to not only code and implement the analysis, but also extract unambiguous timing information from the component model and map it to the inputs for the analysis model. This is because the design and analysis models are often build upon different meta-models [23]. Moreover, the design model can contain redundant timing information. Hence, it is not trivial to extract unambiguous timing information for HRTA and E2EDA.

We divide the timing information (to be extracted) into two categories.

Extraction of Timing Information Corresponding to User Inputs. The first category corresponds to the timing attributes of tasks (in each node) and network messages that are provided in the modeled application by the user. These timing attributes include Worst Case Execution Times (WCETs), periods, minimum update times, offsets, priorities, deadlines, blocking times, precedence relations in task chains, jitters, etc. In [34], we identified all the timing attributes of nodes, networks, transactions, tasks and messages that are required by HRTA. This timing information should be extracted from the modeled application and be made available as an input for the end-to-end response time and delay analysis.

Extraction of Timing Information from the Modeled Application. The second category corresponds to the timing attributes that are not directly provided by the user but they must be extracted from the modeled application. For example, message period (in the case of periodic transmission) or message inhibit time (in the case of sporadic transmission) is often not specified by the user. These attributes must be extracted from the modeled application because they are required by the RTA of network communication. In fact, a message inherits the period or inhibit time from the task that queues it. Thus, we assign period or inhibit time to the message which is equal to the period or inhibit time of its sender task respectively.

However, the extraction of message timing attributes becomes complex when the sender task has both periodic and sporadic activation patterns. In this case, not only the timing attributes of a message have to be extracted but also the transmission type of the message has to be identified. This problem can be visualized in the example shown in Fig. 11. It should be noted that the Out Software Circuit (OSWC), shown in the figure, is one of the network interface

Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin

components in RCM that sends a message to the network. The other network
interface component is In Software Circuit (ISWC) that receives a message from
the network [40].

In Fig. 11(a), the sender task is activated by a clock, and hence the cor-
responding message is periodic. Similarly, the corresponding message is spo-
radic in Fig. 11(b) because the sender task is activated by an event. However,
the sender task in Fig. 11(c) is triggered by both a clock and an event. Here
the relationship between two triggering sources is important. If there exists a
dependency relation between them as in the case of mixed transmission mode
in the CANopen protocol [4] and AUTOSAR communication [9] then such mes-
sage will be considered as a special type of sporadic message. On the other
hand, if triggering sources are independent of each other as in the case of im-
plementation in the HCAN protocol [15] then the corresponding message will
be considered a mixed message [33, 37].

If there are periodic and sporadic messages in the modeled application,
the HRTA plug-in uses the first profile for network analysis (discussed in Sec-
tion 3.3). On the other hand, if the modeled application contains mixed mes-
sages as well, the second profile for network analysis is used. We extract the
transmission type of a message from the modeled application as follows. If the
sender of a message has a periodic or sporadic activation pattern then the mes-
sage is assigned periodic or sporadic transmission type respectively. However,
if the sender is activated periodically as well as sporadically and both trigger-
ing sources are independent of each other, the message is assigned the mixed
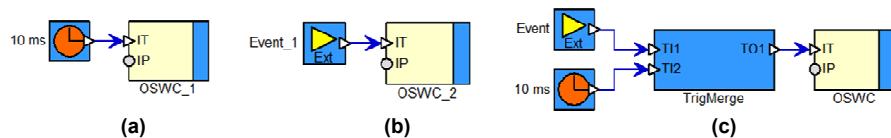transmission type.

(a)   **(b)**   **(c)**

Fig. 11. Extraction of transmission type of a message

Identification of Trigger, Data and Mixed Chains. The end-to-end timing require-
ments on trigger chains are different from those on data chains. These require-
ments correspond to end-to-end response times for trigger chains and both
end-to-end response times and delays for data chains. Data and trigger chains
should be distinctly identified and the corresponding timing requirements should
be unambiguously captured in the timing model on which the analysis tools op-
erate. For this purpose, we add a new attribute "trigger dependency" in the data
structure of tasks in the analysis 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 spec-

ified on this task in the case of dependent triggering. This is because, a task in a trigger chain cannot start its execution before the completion of the previous task in the chain.

However, a system can also be modeled with task chains that are comprised of data chains as well as trigger chains. We call these chains as mixed chains. An example of a mixed chain modeled with RCM is shown in Fig. 12. In this chain, components SWC_A, SWC_B and SWC_E are triggered by independent clocks and which is the property of components in a data chain. Hence, the "trigger dependency" attribute of the tasks corresponding to these three components will be assigned "independent". Whereas, the components SWC_C and SWC_D are triggered by their respective predecessors and which is the property of components in a trigger chain. The "trigger dependency" attribute of the tasks corresponding to these two components will be assigned "dependent".

A task chain is identified by checking the "trigger dependency" parameter for each individual task in the chain. If this parameter is "dependent" for all tasks (except the first or initiating task) then the chain is identified as a trigger chain. On the other hand, if this parameter for each task in the chain is "independent" then the chain is identified as a data chain. However, if this parameter for some tasks is "independent" and for the others it is "dependent" then the chain is considered as mixed.
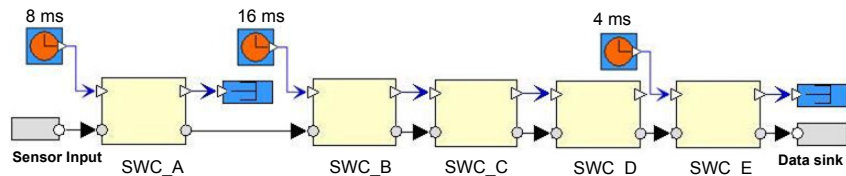
Fig. 12. RCM model of a mixed chain in a single-node real-time system

The problem of identification of a task chain becomes more challenging to resolve when a chain mimics as a data chain as well as a trigger chain by means of trigger merges as shown in Fig. 13. It can be seen that SWC_C component can be triggered by both its predecessor task and a clock. In this case, the "trigger dependency" attribute is assumed to have both the values, i.e., "independent" and "dependent". If such task is identified in a task chain, we consider it as a special type of mixed chain. For this chain, the end-to-end timing requirements correspond to both holistic response times and end-to-end delays.

4.2. Extraction of Linking Information from Distributed Transactions

In order to perform HRTA, correct linking information of DTs should be extracted from the design model [36]. For this purpose, we need to have a mapping
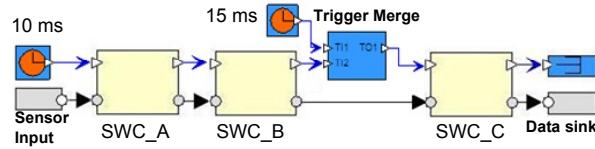
Fig. 13. An example of a data chain with trigger merges

among signals, data ports and messages in the system. Consider the follow-ing DT in a two-node DRE system modeled with RCM as shown in Fig. 14.

$$SWC1 \rightarrow OSWC\_A \rightarrow ISWC\_B \rightarrow SWC2 \rightarrow SWC3$$

In this example, our focus is on the network interface components, i.e., OSWC and ISWC [40]. In order to compute the holistic response time of this DT, we need to extract linking information from the component model. We iden-tified 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 compo-nents 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 DT as shown by the bidirectional arrow in Fig. 14.
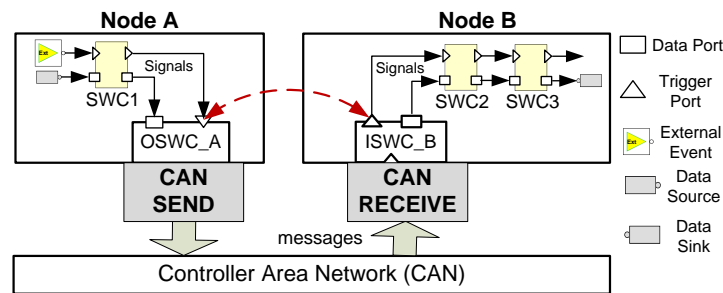
Fig. 14. Two-node DRE system modeled with RCM

Since, the E2EDA plug-in needs to compute all valid timed paths (i.e., those paths in which input actually travels to the output) from initiator to the terminator

for every data chain (see Algorithm 2), the linking information among all tasks and messages in the data chain should be available. We extract this information about all tasks and messages in those data chains on which end-to-end delay constraints are specified. The linking information also includes Trigger Dependency attribute for every task in the chain.

4.3. Analysis of Distributed Transactions with Branches

If a modeled DRE application contains branches of task chains that are distributed over several nodes and have one common terminator task, the calculations for the end-to-end response-time and delays of such chains are not straight forward. Consider the example of a two-node DRE system containing branches in DTs as shown in Fig. 15. The components $OSWC\_A1$ and $OSWC\_A2$ in node A send messages $m1$ and $m2$ which are received by the components $ISWC\_C1$ and $ISWC\_C2$ in node C respectively. Hence, there are two DTs that have different initiators but a single terminator, i.e., SWC_C3. These transactions are listed below.

1. $SWC\_A1 \rightarrow SWC\_A2 \rightarrow OSWC\_A1 \rightarrow ISWC\_C1 \rightarrow SWC\_C1$
   $\rightarrow SWC\_C3$
2. $SWC\_A3 \rightarrow OSWC\_A2 \rightarrow ISWC\_C2 \rightarrow SWC\_C2 \rightarrow SWC\_C3$

Assume that Data Age delay constraint is specified on SWC_C3. Also assume that the start of this constraint is specified on the component SWC_A1 in node A. Therefore, we need to perform end-to-end delay analysis only on the first DT (in the above list). It should be noted that the start (initiating task of the data chain) and end (terminating task of the data chain) of each delay constraint should be specified by the user. We know from Section 3 (Algorithm 2) that the calculations for Data Age delay require the calculation of the holistic response time, i.e., the response time of the last task in the chain (task corresponding to SWC_C3 component). However, the response time of this task depends upon the the holistic response times of both DTs listed above. In this case, the HRTA plug-in will calculate the holistic response times of all branches (two in this case) while the E2EDA plug-in will consider the maximum value among these holistic response times during calculations for the end-to-end delays. Although, the example in Fig. 15 consisted of data chains only, the HRTA plug-in treats trigger chains in a similar fashion.

4.4. Analysis of Mixed Task Chains

The four different end-to-end delays (discussed in Section 3.2) do not exist in the case of trigger chains. This is because trigger chains are analogous to single-rate systems in which over- and under-sampling cannot occur. Moreover, there can never be multiple copies of a single input. If the user specifies end-to-end delay constraints on a trigger chain then the E2EDA plug-in detects this incorrect specification and complains about it. However, a system can also be
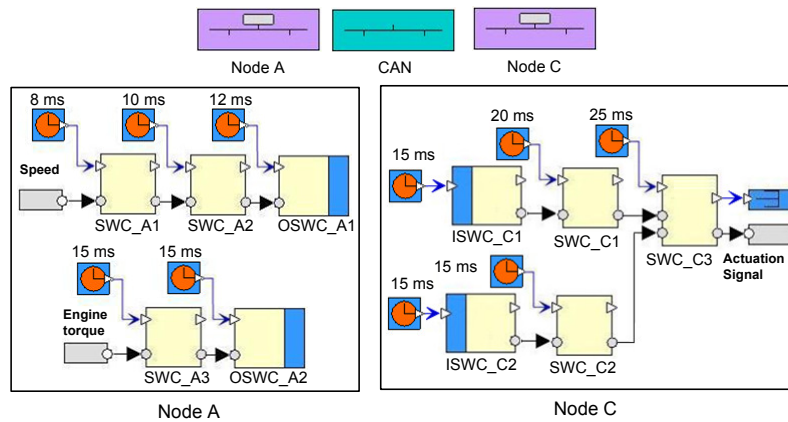
Fig. 15. RCM model of a two-node DRE system with branches in distributed transactions

modeled with mixed chains that are comprised of data chains as well as trigger chains as shown in Fig. 12. Although a mixed chain contains a trigger chain, it is meaningful to compute both holistic response time as well as end-to-end delays for it. Therefore, the newly developed plug-ins compute the holistic response-times as well as end-to-end delays for mixed chains.

There are two options to handle mixed chains in the analysis model. In the first option, if a component is triggered by its predecessor then it is assumed to be triggered by independent clock with the same period as that of its predecessor's clock. Moreover, this component is assumed to have implicit precedence relation with its predecessor, i.e., it can be executed only upon completion of its predecessor's execution. SWC_C and SWC_D are the examples of such components in the mixed chain shown in Fig. 12. Using this option, the execution time line of the task chain corresponding to component chain of Fig. 12 is shown in Fig. 16. This time line will be used by the E2EDA plug-in to compute the total number of timed paths. However, there are several timed paths (indicated with crosses in Fig. 16) that are impossible to occur in reality. This is because each instance of a task in a trigger chain can be triggered only by one instance of its predecessor task. This will result in unnecessary calculations, i.e., a considerable overhead on the execution time of the analysis plug-ins. Therefore, we do not implement this option in the analysis model.

Instead, we use the second option that reduces the mixed chain by combining all tasks belonging to a trigger sub-chain (within the mixed chain) into a single task activated by independent clock. Hence, the reduced mixed chain resembles a data chain. For example, SWC_B, SWC_C and SWC_D are combined to a single task (with combined WCETs, offsets, etc.) which is triggered by independent clock whose period is exactly the same as that of the clock that triggers SWC_B component. The execution time line of the task chain corresponding to reduced mixed chain of Fig. 12 is shown in Fig. 17. The corresponding end-to-end delays are also depicted in Fig. 17. By implementing the
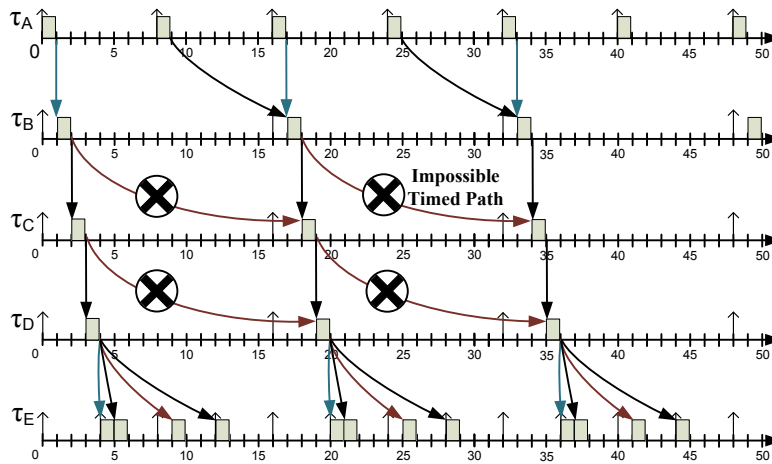
Fig. 16. Demonstration of impossible timed paths in mixed chains

second option , we got rid of the so-called "impossible timed paths". It should be noted that these chain reductions are not required by the HRTA plug-in. Mixed chain reduction method is only applied in the analysis model of the E2EDA plug-in.
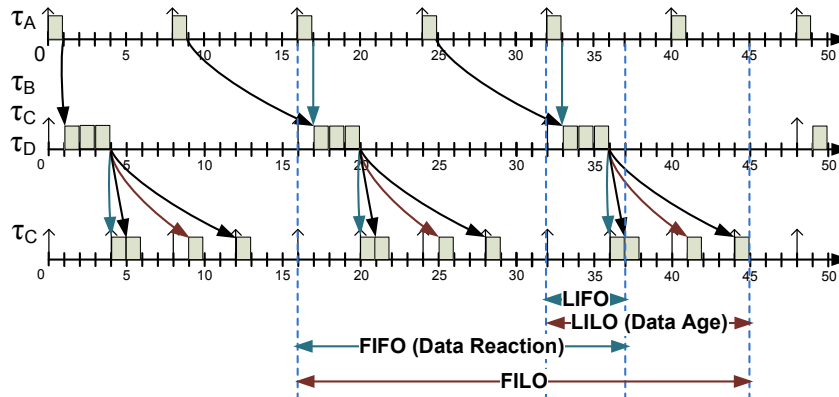

Fig. 17. Reduction of a mixed chain in the analysis model

Mixed chains may also exist in the models of DRE systems where they may contain many combinations of data and trigger chains distributed over several nodes. Four such combinations in a two-node DRE system are shown in Fig. 18. Mixed chain reduction method is applied on distributed mixed chains in a similar fashion.
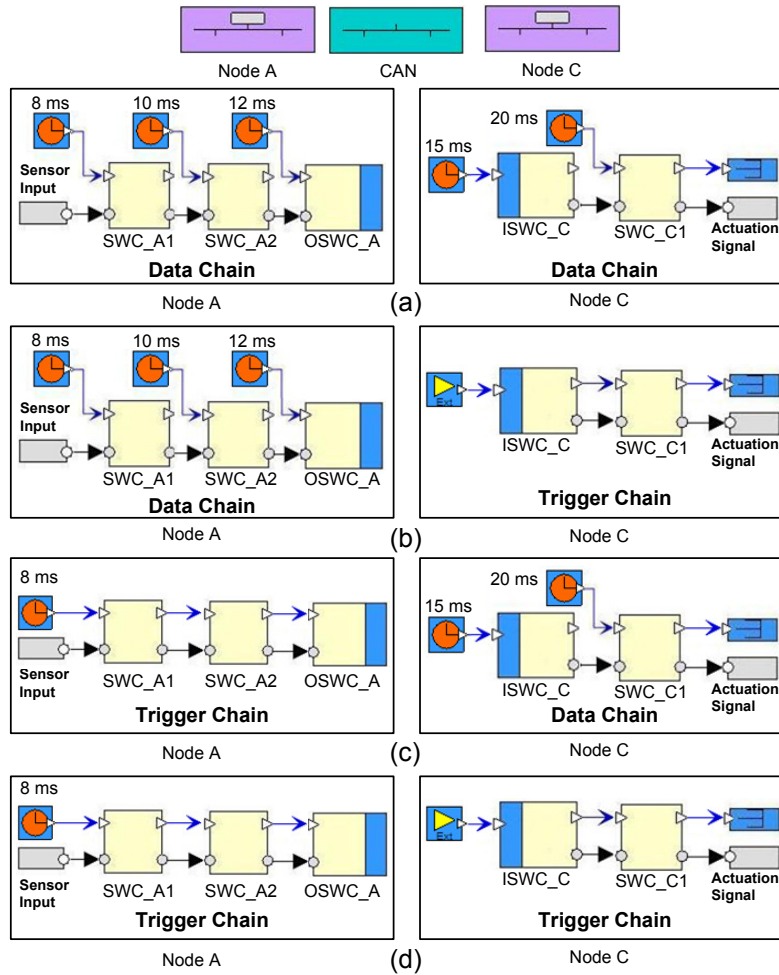
Fig. 18. Different combinations of data and trigger chains in a two-node DRE system modeled with RCM

4.5. Analysis of the System Containing Messages Received from Outside of the Model

One of the requirements by the users of the analysis tools was that the HRTA and E2EDA plug-ins should be able to support the analysis of a system that receives messages from unknown senders (from outside of the modeled application). One motivation behind this requirement may be the integration of two systems that are build using different methodologies and tools. Second motivation could be the integration of legacy systems with newly developed systems. Another motivation could be the requirement for the end-to-end timing analysis early during the development. At early stage, the models of some nodes

may not be available. However, the signals and messages which these missing nodes are supposed to send and receive might have been decided. With these requirements on the system development, the network is assumed to contain messages whose sender nodes are not developed yet. Similarly, the available nodes may send messages via network to the nodes that will be available at a later stage.

As we discussed earlier in Section 3.3, the holistic response-time analysis connects the tasks and messages in a DT by means of attribute inheritance [50]. This means that a message inherits the difference between the worst- and best-case response times of the sending task as its release jitter. Moreover, the message also inherits other attributes from the sender task such as transmission type (periodic, sporadic or mixed [33]); and period or inhibit time or both. Since, the HRTA algorithm is iterative, the attribute inheritance is repeatedly carried out until holistic response time of the chain converges or corresponding deadlines are violated.

The only problem with this requirement is that a message, obviously, cannot inherit these attributes if the sender is unknown or the message is received from outside of the model. In order to solve this problem, we treat all such messages in the analysis model differently from the rest of the messages in the system. Each such message is assumed to be the initiator of the corresponding DT. The transmission type and period (in the case of periodic transmission) or inhibit time (in the case of sporadic transmission) or both (in the case of mixed transmission) [37] of such message are extracted from the user input (instead of the sending task as in the case of intra-model messages). However, the forward attribute inheritance is valid for such messages. This means that the receiver task of this message will inherit the difference between the worst- and best-case response times of the message as its release jitter.

4.6. Impact of Design Decisions in the Component Technology on the Implementation of the Analysis

The design decisions made in the component technology (i.e., RCM) 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 and E2EDA, the implementer 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, their interconnections and software architectures. 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 a chain of components. The run-time translation of a software component may differ among different component models.

4.7. Direct Cycles in Distributed Transactions

A direct cycle in a DT is formed when any two tasks located on different nodes send messages to each other. When there are direct cycles in a DT, the holistic analysis algorithm may run forever and may not produce converging results (if deadlines are not specified), i.e., the response times increase in every iteration.

Consider a two-node application modeled in RCM as shown in Fig. 19 (a). The $OSWC\_A$ component in node A sends a message $m1$ to node B where it is received by $ISWC\_B$ component. Similarly, $OSWC\_B$ component in node B sends a message $m2$ to $ISWC\_A$ component in node A. There are two options for the run-time allocation of network interface components (OSWC and ISWC) as shown in Fig. 19 (b). First option is to allocate a network interface component to the task that corresponds to the immediate SWC, i.e., to the same task as that of the component that receives/sends the signals from/to it. Since SWC_A is immediately connected to both network interface components in node A, there will be only one task in node A denoted by τ_A as shown in Fig. 19 (b). Similarly, τ_B is the run-time representation of $ISWC\_B$, SWC_B and $OSWC\_B$ components. It is obvious that the run-time allocation of network interface components in the first option results in direct cycles. This problem may appear in those component models which do not use exclusive modeling objects or means to differentiate between intra- and inter-node communication in the design model and rely completely on the run-time environment to handle the communication. Hence, some special methods are required to avoid direct cycles in these models.

However, the direct cycles in DTs can be avoided by allocating each network interface component to a separate task as shown in the option 2 in Fig. 19 (b). Although same messages are sent between the nodes, one task cannot be both a sender and a receiver. No doubt, there is a cycle between the nodes, but not a direct one. In this case, the holistic algorithm may produce converging response-times, and non-terminating execution of the plug-in may be avoided. It is interesting to note that the requirements and limitations of the analysis implementation may provide feedback to the design decisions concerning the run-time allocation of modeling components.

4.8. Sequential Execution of Plug-ins in Rubus Plug-in Framework

The plug-in framework in Rubus-ICE allows only sequential execution of plug-ins. Hence, a plug-in has to execute to completion and terminate before the next plug-in can start. It should be noted that there exists a plug-in in Rubus-ICE that can perform RTA of tasks in a node and it is already in the industrial use. There are two options to develop the HRTA plug-in for Rubus-ICE, i.e., option A and B as shown in Fig. 20.

The option A supports reusability by building the HRTA plug-in upon the existing Node RTA Plug-in. Thus, the HRTA plug-in is built by integrating existing RTA plug-in with two new plug-ins, i.e., one implementing network RTA algorithms and the other implementing holistic RTA algorithm. In this case, the

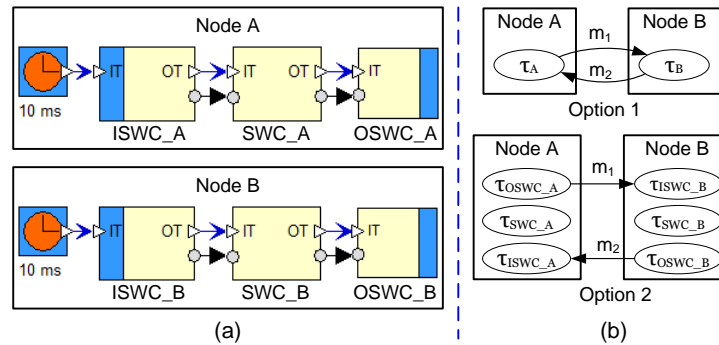Support for end-to-end response-time and delay analysis in the industrial tool

Fig. 19. Options for the run-time allocation of network interface components

HRTA plug-in will be lightweight. 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 or the deadlines (if specified) are violated. On the other hand, option B requires the development of the HRTA plug-in from the scratch, i.e, implementing the algorithms of node, network and holistic RTA. This option does not support any reuse of existing plug-ins.
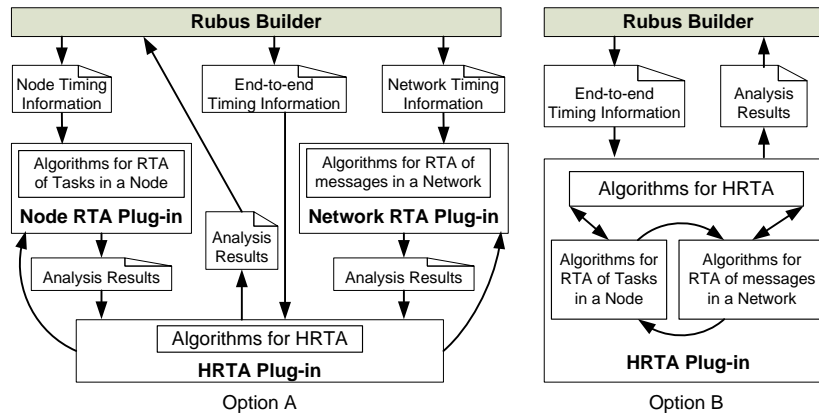
Fig. 20. Options to develop the HRTA Plug-in for Rubus-ICE

Since, option A allows the reuse of a pre-tested and heavyweight (having most complex algorithms compared to the network and holistic RTA) node RTA plug-in, it is easy to implement and requires less time for implementation, integration and 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 executed sequentially. Moreover, one plug-in cannot execute the other. Hence, we selected option B for the implementation of HRTA.

The algorithm for the computation of end-to-end delays requires the response times of all tasks, messages and task chains in the system as one of its inputs. As compared to HRTA algorithm, it is not iterative. Therefore, there is no need to build the E2EDA plug-in from the scratch. In fact, the HRTA plug-in can be completely reused as a black box. This means that the response times of tasks, messages and task chains computed by the HRTA plug-in can be used as one of the inputs for the E2EDA plug-in as shown in Fig. 10.

4.9. Analysis of DRE Systems with Multiple Networks

In a DRE system, a node may be connected to more than one network. This type of node is called a gateway node. If a transaction is distributed over more than one network, the computation of its holistic response time involves the analysis of more than one network. Consider the example of a DRE system with two networks, i.e., CAN and LIN as shown in Fig. 21. There are five nodes in the system. Node 3 is the 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 $Node3$. The computation of holistic response time of this transaction will involve the computation of message response times in both CAN and LIN networks.

If a modeled system contains more than one network, we divide it into subsystems (each having a single network) and analyze them separately in the first step. In the second step, the attribute inheritance is carried out (see Section 3.3) and the subsystems are analyzed again. The second step is repeated until the response times converge or the deadlines (if specified) are violated. In the above example, we first perform HRTA using CAN and LIN networks separately. Then we provide the response times of the messages that are received at the gateway node as input jitters to the receiver tasks (attribute inheritance). Then HRTA of CAN and LIN networks is performed again. These steps are repeated until we get stable response times. Although we analyzed the subsystems separately, the multi-step analysis (especially attribute inheritance step) makes the overall analysis to be holistic.

The implemented HRTA does not support the analysis of a transaction that is distributed cyclically on multiple networks, i.e., the transactions that is distributed over more than one network while its first and last tasks are located on the same network. Since, the E2EDA plug-in receives the response-time results from the HRTA plug-in, it does not need to split the system (with multiple networks) into sub-systems. In fact, the E2EDA plug-in analyzes it as a single system.

4.10. Specification of Delay Constraints on Data Paths

One issue that concerns both modeling and analysis is how to specify the delay constraints on data paths in both data and mixed chains. This is important
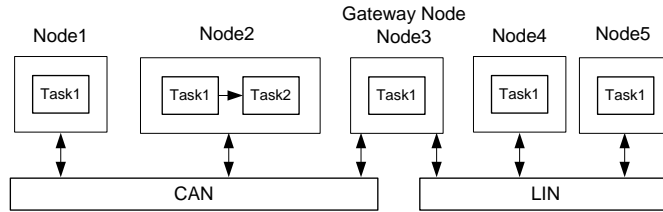
Fig. 21. Multiple networks in a DRE system

because the delay constraints specified in the modeled application have to be extracted in the timing model and the end-to-end delays have to be computed only for the specified data path(s) by the E2EDA plug-in. For this purpose, we introduce start and end objects for each of the four delay constraints (discussed in Subsection 3.2) in the component technology. The constraint object has a meaningful name, and start and end points along a data path. Fig. 22 shows the "Data Age" delay constraint specified on a sensor-actuator data path. Similarly, there are start and end objects for "Data Reaction", "LIFO" and "FILO" delays. All these delay constraints will be used in the case study in Section 6. In the example shown in Fig. 22, the E2EDA plug-in will consider the tasks corresponding to the components sensor_signal_read, filter and compute_actuator_signal while calculating the data age delay. A delay constraint can also be distributed over several nodes. It should be noted that the delay constraints can be specified even on a small segment of a long data path. Another useful method for specifying the delay constraints is by selecting each component (e.g., with mouse click) along the data path. The implementation of this method in Rubus-ICE is left for the future work.
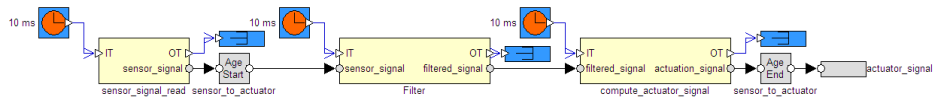

Fig. 22. Age delay constraint specified on a data path

4.11. Presentation of Analysis Results

When HRTA of a modeled application has been performed, the next issue is how to present the analysis results. There can be a large number of tasks and messages in the system. It may not be appropriate to display the response time of all tasks and messages in the system because it may contain a lot of useless information (if the user is not interested in all of it). Furthermore, presenting the end-to-end response times and delays of only DTs to the user may not be appropriate because there may be hundreds of DTs in a DRE application. A

way around this problem is to provide the end-to-end response times and delays of only those tasks and DTs which have deadline requirements and delay constraints (specified by the user) or which produce control signals for external actuators (e.g., the analysis results of case study that will be discussed in Section 6). Apart from this, we also provide an option for the user to get detailed analysis results from both the HRTA and E2EDA plug-ins. The analysis report also shows network utilization which is defined as the sum of the ratio of transmission time to the corresponding period (or minimum-update time) for all messages in the network [33].

4.12. Interaction between the User and the HRTA Plug-in

We identified that it is important to provide a progress report of the HRTA and E2EDA plug-ins during their executions. Based on the progress, the user should be able to interact with the plug-in while it is running. The HRTA algorithm iteratively runs 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 (if deadlines are specified). We feel that it is important to display the number of iterations, running time and over all progress of the plug-in during its execution. Moreover, the user should be able to stop, rerun or exit the plug-in at any time.

4.13. Suggestions to Improve Schedulability Based on Analysis Results

If the analysis results indicate that the modeled system is unschedulable, it can be interesting if the HRTA plug-in is able to provide suggestions (e.g., by varying system parameters) guiding the user to make the system schedulable. However, it is not trivial to provide such feedback because there can be so many reasons behind the system being not schedulable. The support for this type of feedback in the HRTA plug-in will be provided in the future. Another interesting and related feature would be to provide a trace analyzer as another plug-in that can be used after system has been developed. This analyzer will record the execution of the actual system and then present a graphical comparison of the trace with response times of tasks and messages; holistic response times of trigger, data and mixed chains; and end-to-end delays of data and mixed chains. Based on such comparisons, the user may have better understanding of how the schedulability of the system can be improved. The implementation of this feature is left for the future work.

4.14. Requirement for Continuous Collaboration between Integrator and Implementer

Our experience of integrating the HRTA and E2EDA plug-ins with Rubus-ICE shows that there is a need for continuous collaboration between the integrator of the plug-ins and its implementer especially during the phase of integration

testing (see next Section). This collaboration is more obvious when the plug-in is developed in isolation by the implementer (from research background) and integrated with the industrial tool chain by the integrator (with limited experience of integrating complex real-time analysis but aware of overall objective). A continuous consultation and communication was required between the integrator and the implementer for the verification of the plug-ins. Examples of small DRE systems with varying architectures were created for the verification. The implementer had to verify these examples by hand. The integration testing and verification of the HRTA plug-in was non-trivial and most tedious activity.

5. Testing and Evaluation

In this section we discuss our test plan for both standalone and integration testing of the HRTA and E2EDA plug-ins. Error handling and sanity checking routines make 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 the following sets of error handling routines.

- A set of routines evaluating the validity of all inputs: attributes of all nodes, transactions, tasks, networks and messages in the system.
- A set of routines evaluating the validity of linking information of all DTs in the system.
- A set of routines evaluating the validity of intermediate results that are iteratively inherited as inputs (e.g., a message inheriting the difference between the worst- and best-case response times of the sender task as its release jitter).
- A set of routines evaluating the overload conditions during the analysis. For example, processor or network utilization exceeding 100%, and presence of direct cycles in the system. Since HRTA algorithm is iterative, the analysis may never terminate in the presence of these conditions if the deadlines are not specified.
- A set of routines evaluating variable overflow during the analysis.
- A set of routines verifying the design correctness of the modeled application. These routines identify the presence of direct cycles in the modeled application. Moreover, they also identify if the delay constraints are wrongly specified, for example, a delay constraint specified on a trigger chain instead of a data or a mixed chain.

5.1. Standalone Testing

Standalone testing means testing of the implementation of HRTA and E2EDA before they are integrated as plug-ins with the Rubus builder tool. In other words, it refers to the testing of HRTA and E2EDA in isolation. The following input methods were used for the standalone testing.

1. Hard coded input test vectors.

2. Test vectors are read from external files.
3. Test vectors are generated using a test case generator (a separate program). This generator produces test cases with varying architectures. It also randomly inserts invalid inputs to check if the error handling routines are able to catch the errors.

The analysis results provided by the plug-ins corresponding to the test vectors in the first two input methods were also verified by hand.

5.2. Integration Testing

Integration testing refers to the testing of the HRTA and E2EDA plug-ins after they have been integrated with the Rubus builder tool. Although standalone testing is already performed, the integration of these plug-ins with Rubus-ICE may induce unexpected errors. Our experience shows that the integration testing is much more difficult and time consuming activity compared to the standalone testing. The following input methods were used for the integration testing.

1. Test vectors are read from external files.
2. Test vectors are manually written in the ICCM file (see Fig. 1) to make it appear as if test vectors were extracted from the modeled application.
3. Test vectors are automatically extracted from several DRE applications modeled with RCM.

The analysis results provided by the plug-ins corresponding to all types of test cases were also verified by hand.

6. Automotive Application Case Study

We provide a proof of concept for the analysis techniques that we implemented in the Rubus-ICE tool suite by conducting the automotive-application case study. First, we model the Autonomous Cruise Control (ACC) system with RCM using Rubus-ICE. Then, we analyze the modeled ACC system using the HRTA and E2EDA plug-ins.

6.1. Autonomous Cruise Control System

A cruise control system is an automotive feature that allows a vehicle to automatically maintain a steady speed to the value that is preset by the driver. It uses velocity feedback from the speed sensor (e.g., a speedometer) and accordingly controls the engine throttle. However, it does not take into account traffic conditions around the vehicle. Whereas, an Autonomous Cruise Control (ACC) system allows the cruise control of the vehicle to adapt itself to the traffic environment without communicating (cooperating) with the surrounding vehicles. Often, it uses a radar to create a feedback of distance to and velocity of the preceding vehicle. Based on the feedback, it either reduces the vehicle speed

to keep a safe distance and time gap from the preceding vehicle or accelerates the vehicle to match the preset speed specified by the driver [42].

The ACC system may be divided into four subsystems, i.e., Cruise Control, Engine Control, Brake Control and User Interface [14]. Fig. 23 shows the block diagram of the ACC system. The subsystems communicate with each other via the CAN network.

User Interface Subsystem. The User Interface (UI) subsystem reads inputs (provided by the driver) and shows status messages and warnings on the display screen. The inputs are acquired by means of switches and buttons mounted on the steering wheel. These include Cruise Switch input that corresponds to ON/OFF, Standby and Resume (resuming to a speed predefined by the driver) states for ACC; Set Speed input (desired cruising speed set by the driver) and desired clearing distance from the preceding vehicle. Apart from user inputs, it also receives some other parameters from the rest of the subsystems via CAN network. These include linear and angular speed of the vehicle, i.e., kilometer per hour (KPH) and revolution per minute (RPM), status of manual brake sensor, state of ACC subsystem, status messages and warnings to be displayed on the screen. Apart from showing status messages and warnings, it sends messages (including status of driver's input) to other subsystems.
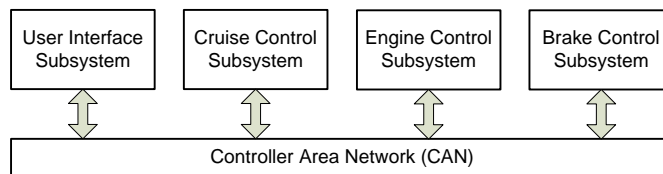
Fig. 23. Block diagram of Autonomous Cruise Control System

Cruise Control Subsystem. The Cruse Control (CC) subsystem receives user input information as a CAN message from the UI subsystem. From the received message it analyzes the state of the cruise control switch; if it is in ON state then it activates the cruise control functionality. It reads input from the proximity sensor (e.g., radar) and processes it to determine the presence of a vehicle in front of it. Moreover, it processes the radar signals along with the information received from other subsystems such as vehicle speed to determine its distance from the preceding vehicle. Accordingly, it sends control information to the Brake Control and Engine Control subsystems to adjust the speed of the vehicle with the cruising speed or clearing distance from the preceding vehicle. It also receives the status of manual brake sensor from the Brake Control subsystem. If brakes are pressed manually then the cruise control functionality is disabled. It also sends status messages to the UI subsystem.

Engine Control Subsystem. The Engine Control (EC) subsystem is responsible for controlling the vehicle speed by adjusting engine throttle. It reads sensor input and accordingly determines engine torque. It receives CAN messages sent by other subsystems. The messages include information regarding vehicle speed, status of manual brake sensor, and input information processed by the UI system. Based on the received information, it determines whether to increase or decrease engine throttle. It then sends new throttle position to the actuators that control engine throttle.

Brake Control Subsystem. The Brake Control (BC) subsystem receives inputs from sensor for manual brakes status and linear and angular speed sensors connected to all wheels. It also receives a CAN message that includes control information processed by the CC subsystem. Based on this feedback, it computes new vehicle speed. Accordingly, it produces control signals and sends them to the brake actuators and brake light controllers. It also sends CAN messages to other subsystems that carry information regarding status of manual brake, vehicle speed and RPM.

6.2. Modeling of ACC System with RCM in Rubus-ICE

In RCM, we model each subsystem as a separate node connected to a CAN network as shown in Fig. 24. The selected speed of the CAN bus is 500 kbps. The extended frame format is selected which means that each frame will use 29-bit identifier [26]. The ACC system is modeled with trigger, data and mixed chains.
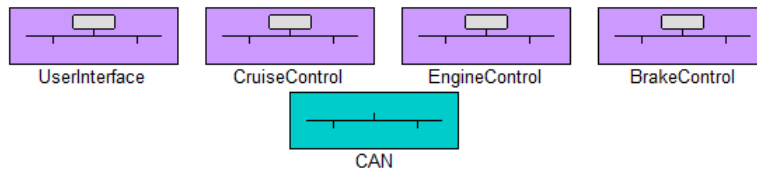
Fig. 24. Autonomous Cruise Control System modeled with RCM

There are seven CAN messages that are sent by the nodes as shown in Fig. 25. A signal data base "signalDB" that contains all the signals sent to the network is also shown. Each signal in the signal database is linked to one or more messages. The extracted attributes of all messages including data size (s_m), priority (P_m), transmission type (ξ_m) and period or minimum inter-arrival time (T_m) are listed in Table 1.

The high-level architectures of CC, EC, BC and UI nodes modeled with RCM are shown in Fig. 26, 27, 28 and 29 respectively.
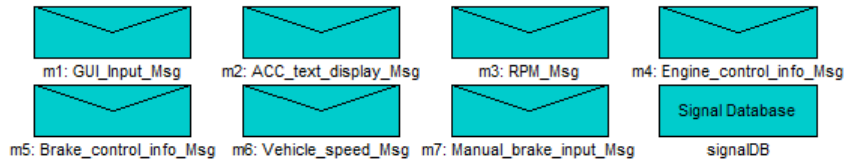
Fig. 25. CAN messages and signal database modeled with RCM

Table 1. Message attributes extracted from the model

Msg	s_m	P_m	ξ_m	T_m **(μSec)**	C_m **(μSec)**
$m1$	8	7	Periodic	10000	320
$m2$	8	6	Periodic	10000	320
$m3$	8	4	Periodic	10000	200
$m4$	8	3	Sporadic	10000	320
$m5$	2	5	Sporadic	10000	320
$m6$	2	2	Periodic	10000	200
$m7$	1	1	Sporadic	10000	180
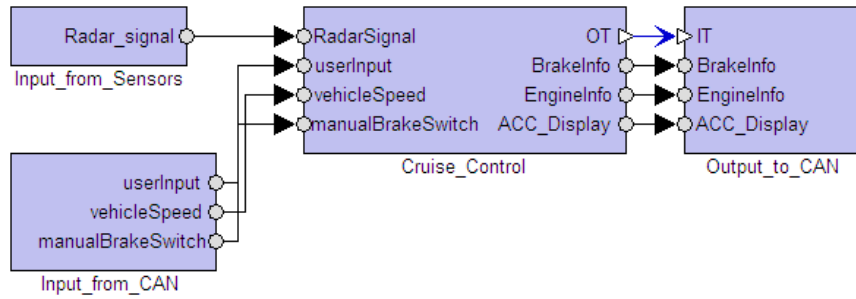
Fig. 26. RCM model of the Cruise Control node
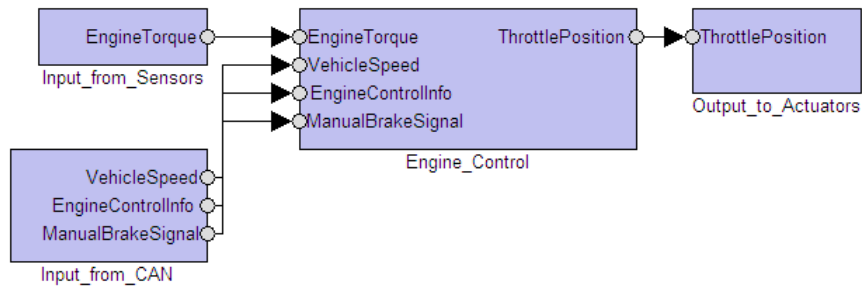
Fig. 27. RCM model of the Engine Control node
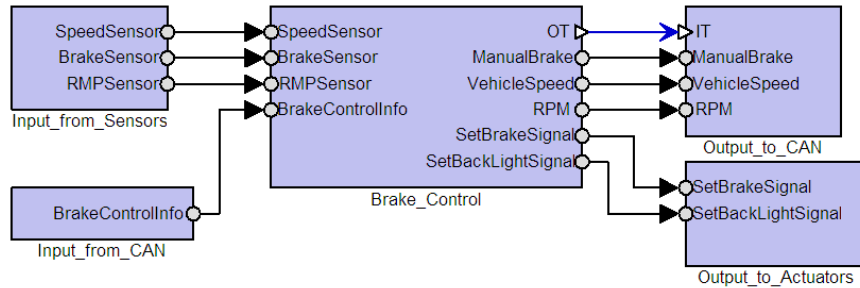
Fig. 28. RCM model of the Brake Control node
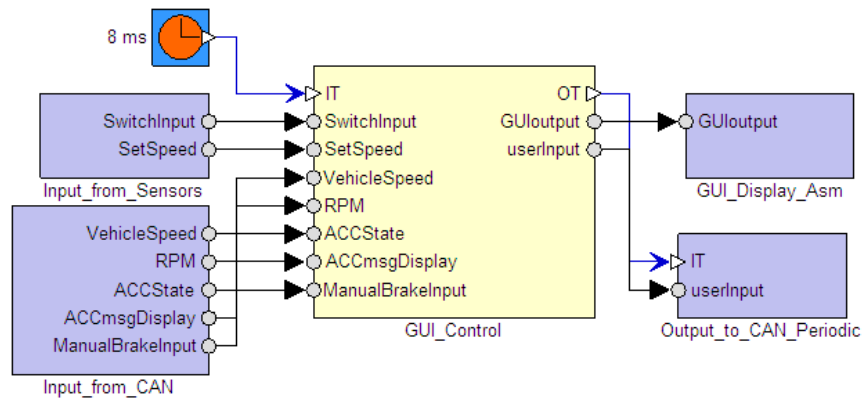
Fig. 29. RCM model of the User Interface node

Internal Model of Cruise Control Node in RCM. The CC node is modeled with four assemblies as shown in Fig. 26. An assembly in RCM is a container for various software items. The Input_from_Sensors assembly contains one SWC that reads radar sensor values as shown in Fig. 30. The Input_from_CAN assembly contains three ISWCs, i.e., GUI_Input_Msg_ISWC, Vehicle_speed_Msg_ISWC and Manual_brake_input_Msg_ISWC as depicted in Fig. 31. These components receive messages $m1$, $m6$ and $m7$ from the CAN network respectively. Similarly, the assembly Output_to_CAN contains three OSWC components as shown in Fig. 32. These components send messages $m5$, $m4$ and $m2$ to the CAN network. The Cruise_Control assembly contains two SWCs: one handles the input and cruise control mode signals while the other processes the received information and produces control messages for the other nodes. The internal model of this assembly is shown in Fig. 33.

Internal Model of Engine Control Node in RCM. The Engine Control node is modeled with four assemblies as shown in Fig. 27. The Input_from_Sensors assembly contains one SWC that reads the sensor values corresponding to
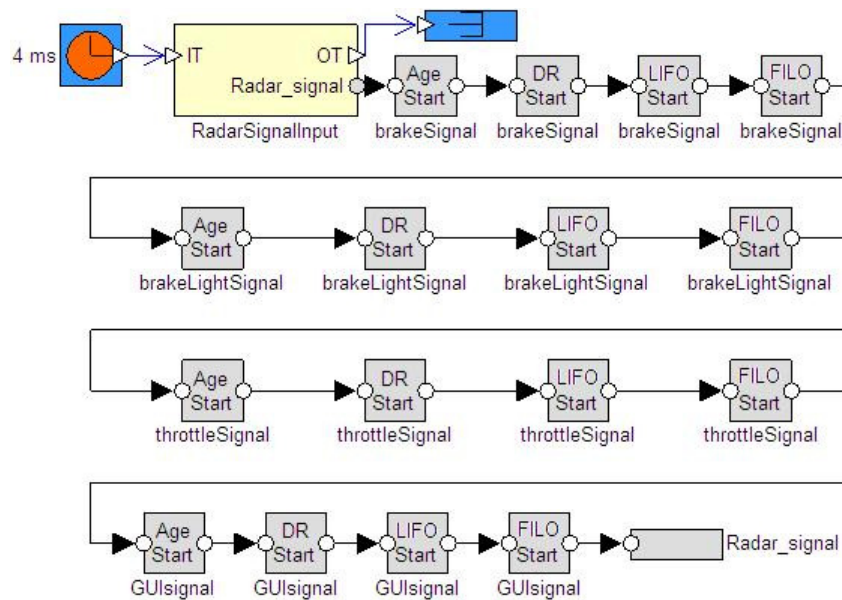
Fig. 30. CC node: Internal model of the Input_from_Sensors assembly
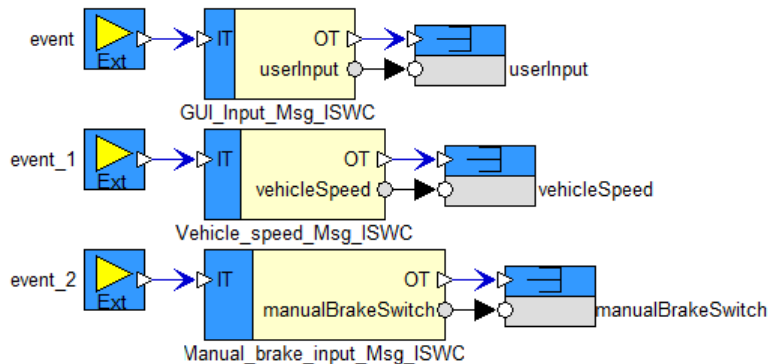

Fig. 31. CC node: Internal model of the Input_from_CAN assembly

the engine torque as shown in Fig. 34. The Input_from_CAN assembly contains three ISWCs, i.e., Vehicle_Speed_Msg_ISWC, Engine_control_info_Msg _ISWC and Manual_brake_input_Msg_ISWC as shown in Fig. 35. These components receive messages $m6$, $m4$ and $m7$ from the CAN network respectively. The third assembly, Output_to_Actuators as shown in Fig. 36, contains the SWC that produces control signals for the engine throttle actuator. The fourth assembly, i.e., Engine_Control as shown in Fig. 37, contains two SWCs: one handles and processes the inputs from sensors and received messages, while the other computes the new position for the engine throttle. These components are part
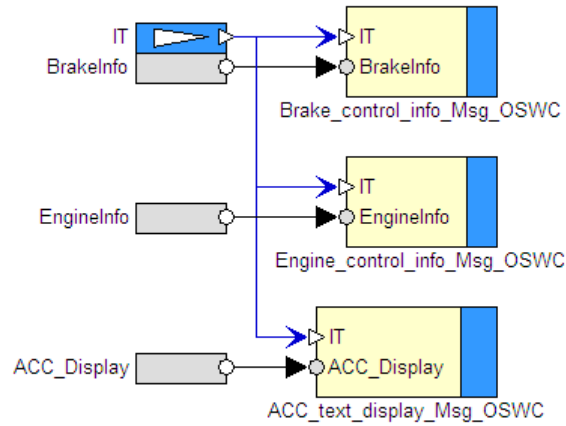
Fig. 32. CC node: Internal model of the Output_to_CAN assembly
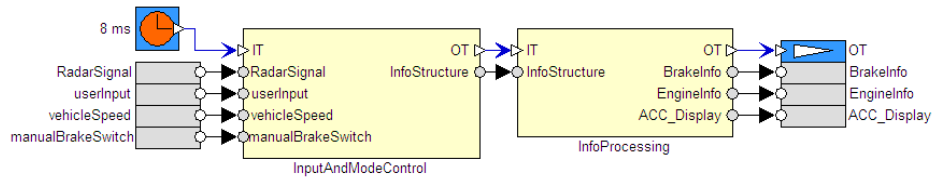

Fig. 33. CC node: SWCs comprising the Cruise_Control assembly

of a distributed mixed chain that we will analyze along with other distributed mixed chains in the next subsections.
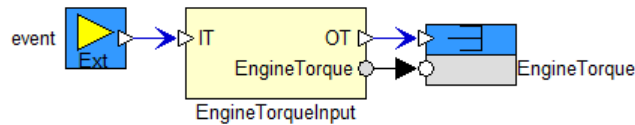

Fig. 34. EC node: Internal model of the Input_from_Sensors assembly

Internal Model of Brake Control Node in RCM. The Brake Control node is modeled with five assemblies as shown in Fig. 28. The Input_from_Sensors assembly contains three SWCs as shown in Fig. 38. These SWCs read the sensor values that correspond to the values of speed, rpm and manual brake sensors in the vehicle. The Input_from_CAN assembly, shown in Fig. 39, contains the ISWC component Brake_control_info_Msg_ISWC that receives a message $m5$ from the CAN network.
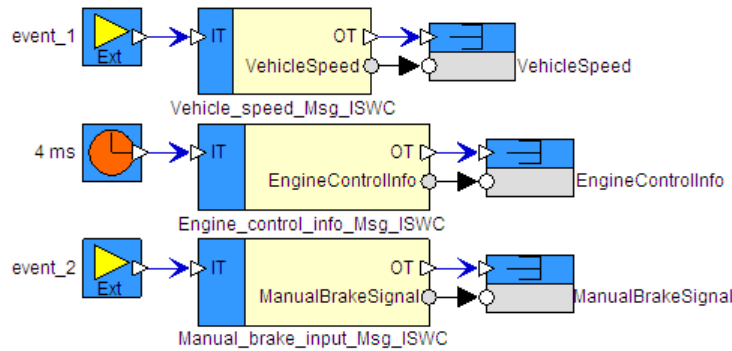
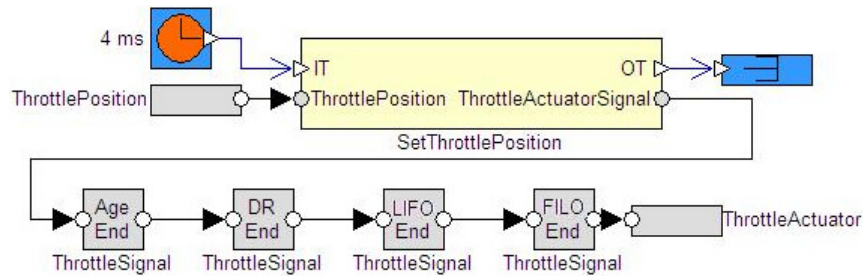**Fig. 35.** EC node: Internal model of the Input_from_CAN assembly

Fig. 36. EC node: Internal model of the Output_to_Actuators assembly
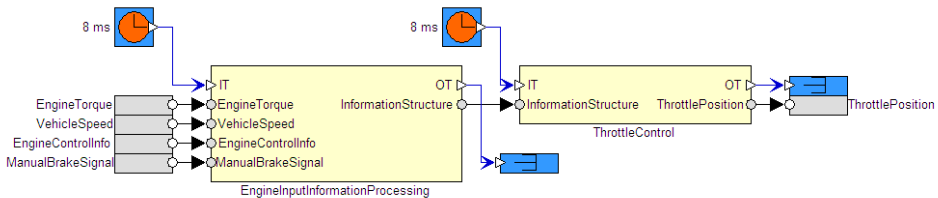

Fig. 37. EC node: SWCs comprising the Engine_Control assembly

The third assembly, i.e., Brake_Control as shown in Fig. 40, contains two SWCs: one handles and processes the inputs from sensors and received messages while the other computes the control signals for brake actuators. The fourth assembly Output_to_CAN contains three OSWC components as shown in Fig. 41. These components send messages $m7$, $m6$ and $m3$ to the CAN network. The fifth assembly, Output_to_Actuators as shown in Fig. 42, contains the SWCs that produce control signals for the brake actuators and brake light controllers.
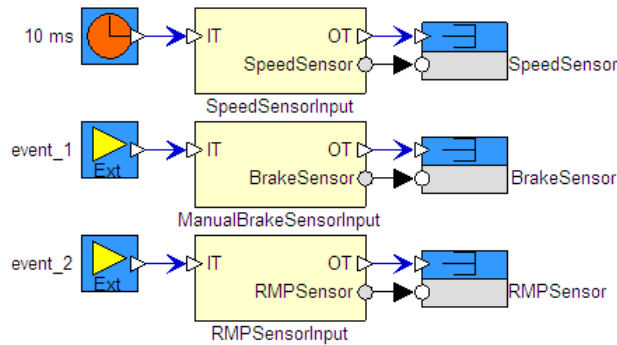
Fig. 38. BC node: Internal model of the Input_from_Sensors assembly

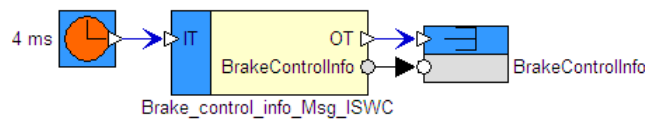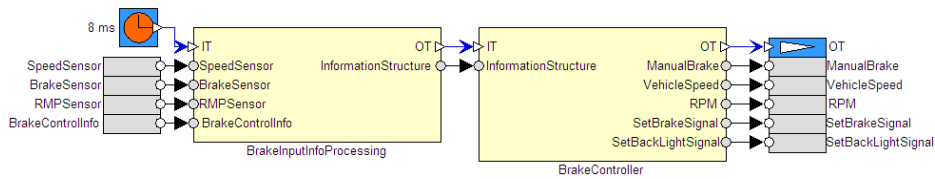**Fig. 39.** BC node: Internal model of the Input_from_CAN assembly

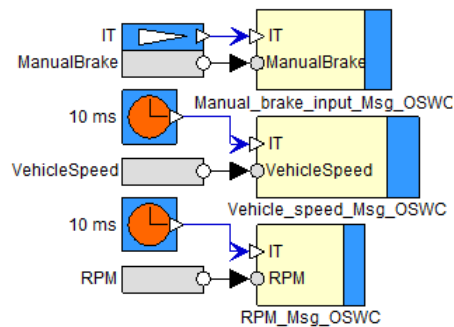**Fig. 40.** BC node: Internal model of the Brake_Control assembly

Fig. 41. BC node: Internal model of the Output_to_CAN assembly

Internal Model of User Interface Node in RCM. The User Interface node is modeled with four assemblies along with one SWC as shown in Fig. 29. The GUI_Control SWC handles the input from the sensors and messages from
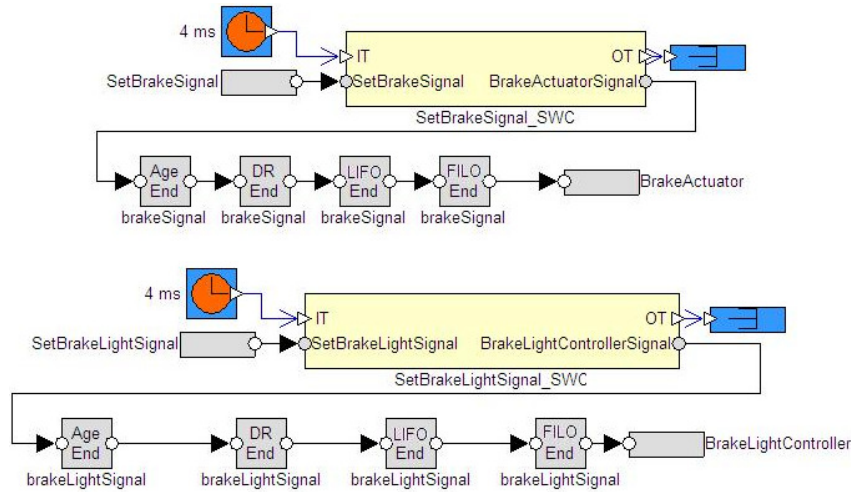
Fig. 42. BC node: Internal model of the Output_to_Actuators assembly

the CAN network. After processing the information, it not only produces information for Graphical User Interface (GUI), but also computes control signals for the other nodes. The Input_from_Sensors assembly contains two SWCs as shown in Fig. 43. One of them reads the sensor values that correspond to the state of the cruise control switch on the steering wheel. The other SWC reads the sensor values that correspond to the vehicle cruising speed set by the driver. The Input_from_CAN assembly contains four ISWC components, i.e., Vehicle_Speed_Msg_ISWC, RPM_Msg_ISWC, Manual_brake_input_Msg_ISWC and ACC_text_display_Msg_ISWC as shown in Fig. 44. These components receive messages $m6$, $m3$, $m7$ and $m2$ from the CAN network respectively. The third assembly, i.e., Output_to_CAN_Periodic sends a message $m1$ to the CAN network via the OSWC component as shown in Fig. 45. The fourth assembly, i.e., GUI_Display_Asm contains one SWC, i.e., GUIdisplay component as shown in Fig. 46. This component sends the signals (corresponding to updated information) to GUI in the car.
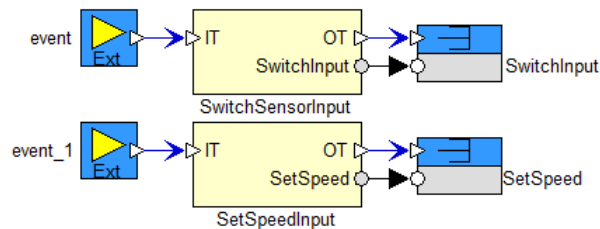

Fig. 43. UI node: Internal model of the Input_from_Sensors assembly
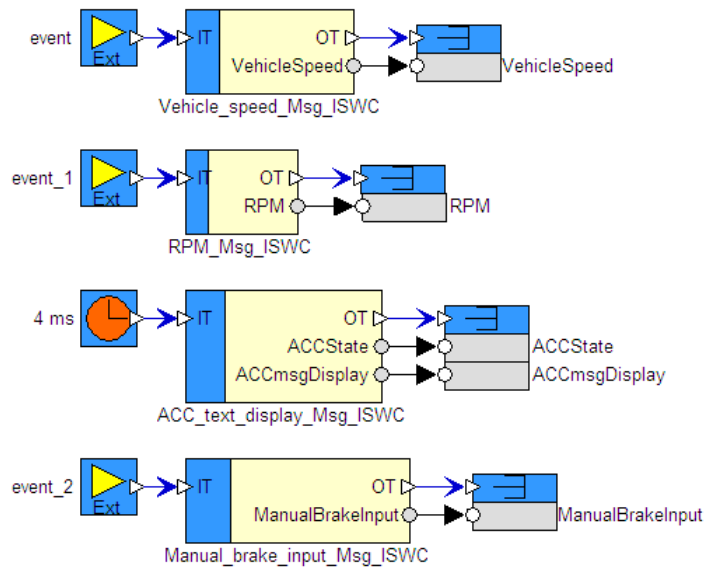
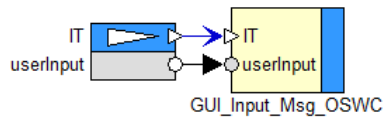**Fig. 44.** UI node: Internal model of the Input_from_CAN assembly

Fig. 45. UI node: Internal model of the Output_to_CAN_Periodic assembly
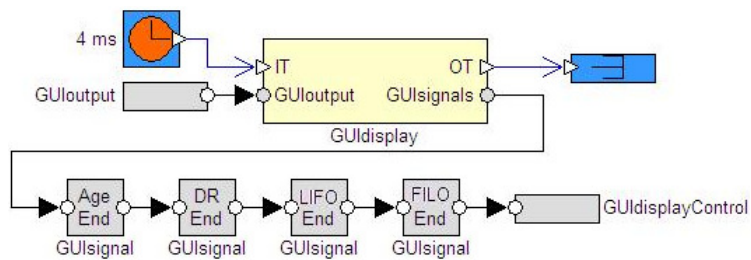

Fig. 46. UI node: Internal model of the GUI_Display_Asm assembly

6.3. Modeling of End-to-end Deadline Requirements

We specify end-to-end deadline requirements on four DTs in the ACC system using a deadline object in RCM. All these DTs, i.e., DT_1, DT_2, DT_3 and DT_4 are distributed mixed chains as shown in Table 2. All these chains have one common initiator, i.e., their first task corresponds to the SWC that reads radar signal which is denoted by *RadarSignalInput* and located in the Cruise Control node as shown in Fig. 30. The last tasks of DT_1 and DT_2 are located in the Brake Control node. These tasks correspond to the SWCs *SetBrakeSignal* and *SetBrakeLightSignal* as shown in Fig. 28. These two tasks are responsible for producing brake actuation and brake light control signals respectively. The last task of DT_3 corresponds to *SetThrottlePosition* SWC and is located in the Engine Control node as shown in Fig. 27. This task is responsible for producing control signal for the engine throttle actuator. The last task of DT_4 corresponds to *GUIdisplay* SWC and is located in the User Interface node as shown in Fig. 29. This task is responsible for providing display information for the driver.

    All the mixed chains under analysis are distributed over more than one node. For the sake of convenience, we list all the components in the data path (from initiator to terminator) of each chain as shown below. We also specify four delay constraints (discussed in Section 3) on each DT under analysis. In RCM, the model of each delay constraint consists of start object and end object. The start objects for all four delay constraints for each DT are shown in Fig. 30. There are sixteen start objects for delay constraints in Fig. 30 because there are four DTs under analysis with four delay constraints specified on each DT. The end objects for all delay constraints for DT_1 and DT_2 are specified in Fig. 42. Similarly, the end objects for all delay constraints for DT_3 and DT_4 are specified in Fig. 36 and Fig. 46 respectively.

1. DT_1: $RadarSignalInput \rightarrow InputAndModeControl \rightarrow InfoProcessing \rightarrow Brake\_control\_info\_Msg\_OSWC \rightarrow message : m5 \rightarrow Brake\_control\_info\_Msg\_ISWC \rightarrow BrakeInputInfoProcessing \rightarrow BrakeController \rightarrow SetBrakeSignal\_SWC$

2. DT_2: $RadarSignalInput \rightarrow InputAndModeControl \rightarrow InfoProcessing \rightarrow Brake\_control\_info\_Msg\_OSWC \rightarrow message : m5 \rightarrow Brake\_control\_info\_Msg\_ISWC \rightarrow BrakeInputInfoProcessing \rightarrow BrakeController \rightarrow SetBrakeLightSignal\_SWC$

3. DT_3: $RadarSignalInput \rightarrow InputAndModeControl \rightarrow InfoProcessing \rightarrow Engine\_control\_info\_Msg\_OSWC \rightarrow message : m4 \rightarrow Engine\_control\_info\_Msg\_ISWC \rightarrow EngineInputInformationProcessing \rightarrow ThrottleControl \rightarrow SetThrottlePosition$

4. DT_4: $RadarSignalInput \rightarrow InputAndModeControl \rightarrow InfoProcessing \rightarrow ACC\_text\_display\_Msg\_OSWC \rightarrow message : m2 \rightarrow ACC\_text\_display\_Msg\_ISWC \rightarrow GUI\_Control \rightarrow GUIdisplay$

6.4. Analysis of ACC System using the HRTA and E2EDA Plug-ins

The run-time allocation of all the components in the model of the ACC system results in 19 transactions, 36 tasks and 7 messages. We provide the analysis results of only those transactions on which deadline requirements or delay constraints are specified. The transmission times (C_m) of all messages computed by the HRTA plug-in are shown in Table 1. The WCET of each component in the modeled ACC system is selected from the range of 10-60 μSec. The HRTA plug-in analyzes all four DTs (discussed in the previous subsection). Once the HRTA plug-in has completed its execution and produced analysis results then the E2EDA plug-in analyzes only those DTs on which end-to-end delay constraints are specified (i.e., all four DTs).

The analysis report in Table 2 provides worst-case holistic response times of the four distributed mixed chains using the HRTA plug-in. The corresponding deadlines are also shown. The response time of a DT is counted from the activation of the first task to the completion of the last task in the chain. The response times of these four DTs correspond to the production of control signals for brake actuators, brake lights controllers, engine throttle actuator and graphical user interface.

The analysis report produced by the E2EDA plug-in is shown in Table 3. It lists four end-to-end delays calculated for each DT under analysis. The corresponding specified delay constraints are also listed in the table. By comparing the end-to-end deadlines and specified delay constraints with the calculated holistic response times and end-to-end delays in Tables 2 and 3 respectively, we see that the modeled ACC system meets all of its deadlines.

Table 2. Analysis report by the HRTA plug-in

Distributed Transaction	Chain Type	Control Signal Produced by the Chain	Deadline (μSec)	Holistic Response Time (μSec)
DT_1	Mixed Chain	SetBrakeSignal	1000	220
DT_2	Mixed Chain	SetBrakeLightSignal	1000	280
DT_3	Mixed Chain	SetThrottlePosition	1000	130
DT_4	Mixed Chain	GUIdisplay	1500	345

7. Conclusion and Future Work

We presented the implementation of the state-of-the-art Holistic Response Time Analysis (HRTA) and End-to-End Delay Analysis (E2EDA) as two individual plug-ins for the existing industrial tool suite Rubus-ICE. The implemented analyses are general as they support the integration of real-time analysis of various networks without a need for changing the end-to-end analysis algorithms. With

Table 3. Analysis report by the E2EDA plug-in

Distributed Transaction	DT_1	DT_2	DT_3	DT_4
Specified Age Delay Constraint(μSec)	5000	5000	5000	5000
Calculated Age Delay (μSec)	4220	4280	4130	4345
Specified Reaction Delay Constraint(μSec)	10000	10000	10000	10000
Calculated Reaction Delay (μSec)	8220	8280	8130	8345
Specified LIFO Delay Constraint(μSec)	1000	1000	1000	1500
Calculated LIFO Delay (μSec)	220	280	130	345
Specified FILO Delay Constraint(μSec)	15000	15000	15000	15000
Calculated FILO Delay (μSec)	12220	12280	12130	12345

the implementation of these plug-ins, Rubus-ICE is able to support distributed end-to-end timing analysis of trigger flows as well as asynchronous data flows which are common in automotive embedded systems.

There are many challenges faced by the implementer when state-of-the-art real-time analyses like HRTA and E2EDA are transferred to the industrial tools. The implementer has to not only code and implement the analyses in the tools, but also deal with various challenging issues in an effective way with respect to time and cost. We discussed and solved several issues that we faced during the implementation, integration and evaluation of the HRTA and E2EDA plug-ins. The experience gained by dealing with the implementation challenges provided a feed back to the component technology (i.e., the Rubus Component Model), for example, feed back on the design decisions for efficient run-time allocation of network interface components.

We also discussed the steps that we followed for testing and evaluating the HRTA and E2EDA plug-ins. We found the integration testing to be a tedious and non-trivial activity. Our experience of implementing, integrating and evaluating these plug-ins shows that a considerable amount of work and time is required to transfer complex real-time analysis results to the industrial tools.

We provided a proof of concept by modeling the autonomous cruise control system with component-based development approach using the existing industrial component model (Rubus Component Model) and analyzing it with the HRTA and E2EDA plug-ins.

We believe that most of the problems discussed in this paper are generally applicable when real-time analysis is transferred to any industrial or academic tool suite. Moreover, the contributions in this paper may provide guidance for the implementation of other complex real-time analysis techniques in any industrial tool suite that supports a plug-in framework for the integration of new tools and allows component-based development of distributed real-time embedded systems.

In the future, we plan to implement the analysis of other network communication protocols (e.g., Flexray, switched ethernet, etc.) and integrate them within the HRTA plug-in. Another future work is the implementation of RTA for CAN

Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin

with FIFO and work-conserving queues [18, 20], and RTA of CAN with FIFO Queues for Mixed Messages [37] within HRTA plug-in. We also plan to integrate the stand alone analyzer, that we developed for the analysis of mixed messages with offsets [39], with the HRTA plug-in.

Acknowledgement

References

1. Arcticus Systems, http://www.arcticus-systems.com
2. BAE Systems Hägglunds, http://www.baesystems.com/hagglunds
3. CANoe. http://www.vector.com/portal/medien/cmc/info/canoe_productinformation_en.pdf
4. CANopen Application Layer and Communication Profile. CiA Draft Standard 301. Version 4.02. February 13, 2002, http://www.can-cia.org/index.php?id=440
5. Knorr-bremse, web page, http://www.knorr-bremse.com
6. MAST–Modeling and Analysis Suite for Real-Time Applications, http://mast.uni-can.es
7. Mecel, web page, http://www.mecel.se
8. RAPID RMA: The Art of Modeling Real-Time Systems, http://www.tripac.com/rapid-rma
9. Requirements on Communication, Release 3.0, Revision 7, Ver. 2.2.0. The AUTOSAR Consortium, September, 2010, www.autosar.org/download/R3.0/AUTO-SAR_SRS_COM.pdf
10. The Volcano Family, http://www.mentor.com/products/vnd
11. Vector. http://www.vector.com
12. Volcano Network Architect. Mentor Graphics, http://www.mentor.com/products/vnd/communication-management/vna
13. Volvo Construction Equipment, http://www.volvoce.com
14. Adaptive Cruise Control System Overview. In: Workshop of Software System Safety Working Group (April 2005), Anaheim, California, USA. Available at: sunnyday.mit.edu/Adaptive_Cruise_Control_Sys_Overview.pdf
15. Hägglunds Controller Area Network (HCAN), Network Implementation Specification. BAE Systems Hägglunds, Sweden (internal document) (April 2009)
16. TIMMO Methodology , Version 2. TIMMO (TIMing MOdel), Deliverable 7 (October 2009), The TIMMO Consortium
17. Audsley, N., Burns, A., Davis, R., Tindell, K., Wellings, A.: Fixed priority pre-emptive scheduling:an historic perspective. Real-Time Systems 8(2/3), 173–198 (1995)
18. Davis, R., Navet, N.: Controller Area Network (CAN) Schedulability Analysis for Messages with Arbitrary Deadlines in FIFO and Work-Conserving Queues. In: 9th IEEE International Workshop on Factory Communication Systems (WFCS). pp. 33 –42 (May 2012)

19. Davis, R., Burns, A., Bril, R., Lukkien, J.: Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised. Real-Time Systems 35, 239–272 (2007)
20. Davis, R.I., Kollmann, S., Pollex, V., Slomka, F.: Controller Area Network (CAN) Schedulability Analysis with FIFO queues. In: 23rd Euromicro Conference on Real-Time Systems (July 2011)
21. Feiertag, N., Richter, K., Nordlander, J., Jonsson, J.: A Compositional Framework for End-to-End Path Delay Calculation of Automotive Systems under Different Path Semantics. In: Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS) (December 2008)
22. Hagner, M., Goltz, U.: Integration of scheduling analysis into uml based development processes through model transformation. In: International Multi-conference on Computer Science and Information Technology (IMCSIT). pp. 797 –804 (October 2010)
23. Hagner, M., Goltz, U.: Integration of scheduling analysis into uml based development processes through model transformation. In: International Multi-conference on Computer Science and Information Technology (IMCSIT). pp. 797 –804 (October 2010)
24. Hamann, A., Henia, R., Racu, R., Jersak, M., Richter, K., Ernst, R.: Symta/s - symbolic timing analysis for systems (2004)
25. Hänninen et.al., K.: The Rubus Component Model for Resource Constrained Real-Time Systems. In: 3rd IEEE International Symposium on Industrial Embedded Systems (June 2008)
26. ISO 11898-1: Road Vehicles interchange of digital information controller area network (CAN) for high-speed communication, ISO Standard-11898, Nov. 1993.
27. Joseph, M., Pandya, P.: Finding Response Times in a Real-Time System. The Computer Journal (British Computer Society) 29(5), 390–395 (October 1986)
28. K. Hänninen et.al.: Framework for real-time analysis in Rubus-ICE. In: 13th IEEE Conference on Emerging Technologies and Factory Automation (ETFA). pp. 782 –788 (2008)
29. Liu, C., Layland, J.: Scheduling algorithms for multi-programming in a hard-real-time environment. ACM 20(1), 46–61 (1973)
30. Mäki-Turja, J., , Nolin, M.: Tighter response-times for tasks with offsets. In: Real-time and Embedded Computing Systems and Applications Conference (RTCSA) (August 2004)
31. Mubeen, S.: Modeling and timing analysis of industrial component-based distributed real-time embedded systems. Licentiate thesis, Mälardalen University (January 2012), http://www.mrtc.mdh.se/index.php?choice=publications&id=2748
32. Mubeen, S., Mäki-Turja, J., Sjödin, M.: Extending response-time analysis of controller area network (CAN) with FIFO queues for mixed messages. In: 16th IEEE Conference on Emerging Technologies and Factory Automation (ETFA). pp. 1–4 (September 2011)
33. Mubeen, S., Mäki-Turja, J., Sjödin, M.: Extending schedulability analysis of controller area network (CAN) for mixed (periodic/sporadic) messages. In: 16th IEEE Conference on Emerging Technologies and Factory Automation (ETFA) (September 2011)
34. Mubeen, S., Mäki-Turja, J., Sjödin, M.: 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. pp. 1–6. Springer (October 2011)

35. Mubeen, S., Mäki-Turja, J., Sjödin, M.: Implementation of Holistic Response-Time Analysis in Rubus-ICE: Preliminary Findings, Issues and Experiences. In: The 32nd IEEE Real-Time Systems Symposium (RTSS), WIP Session. pp. 9–12 (December 2011)

36. Mubeen, S., Mäki-Turja, J., Sjödin, M.: Tracing event chains for holistic response-time analysis of component-based distributed real-time systems. SIGBED Review 8, 48–51 (September 2011), http://doi.acm.org/10.1145/2038617.2038628

37. Mubeen, S., Mäki-Turja, J., Sjödin, M.: Response-Time Analysis of Mixed Messages in Controller Area Network with Priority- and FIFO-Queued Nodes. In: 9th IEEE International Workshop on Factory Communication Systems (WFCS) (May 2012)

38. Mubeen, S., Mäki-Turja, J., Sjödin, M.: Support for Holistic Response-time Analysis in an Industrial Tool Suite: Implementation Issues, Experiences and a Case Study. In: 19th IEEE Conference on Engineering of Computer Based Systems (ECBS). pp. 210 –221 (April 2012)

39. Mubeen, S., Mäki-Turja, J., Sjödin, M.: Worst-case response-time analysis for mixed messages with offsets in controller area network. In: 17th IEEE Conference on Emerging Technologies and Factory Automation (ETFA) (September 2012)

40. Mubeen, S., Mäki-Turja, J., Sjödin, M., Carlson, J.: Analyzable modeling of legacy communication in component-based distributed embedded systems. In: 37th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). pp. 229–238 (September 2011)

41. Nolin, M., Mäki-Turja, J., Hänninen, K.: Achieving Industrial Strength Timing Predictions of Embedded System Behavior. In: ESA. pp. 173–178 (2008)

42. P. Berggren: Autonomous Cruise Control for Chalmers Vehicle Simulator. Master's thesis, Department of Signals and Systems, Chalmers University of Technology (2008)

43. Palencia, J., Harbour, M.G.: Schedulability Analysis for Tasks with Static and Dynamic Offsets. IEEE International Symposium on Real-Time Systems p. 26 (1998)

44. Pop, T., Eles, P., Peng, Z.: Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems. In: Proceedings of the tenth international symposium on Hardware/software codesign. pp. 187–192. CODES '02, ACM, New York, USA (2002)

45. Rajeev, A.C., Mohalik, S., Dixit, M.G., Chokshi, D.B., Ramesh, S.: Schedulability and end-to-end latency in distributed ecu networks: formal modeling and precise estimation. In: Proceedings of the tenth ACM international conference on Embedded software. pp. 129–138. EMSOFT '10, ACM (2010)

46. Schmidt, D., Kuhns, F.: An overview of the Real-Time CORBA specification. Computer 33(6), 56 –63 (June 2000)

47. Sha, L., Abdelzaher, T., rzén, K.E.A., Cervin, A., Baker, T.P., Burns, A., Buttazzo, G., Caccamo, M., Lehoczky, J.P., Mok, A.K.: Real Time Scheduling Theory: A Historical Perspective. Real-Time Systems 28(2/3), 101–155 (2004)

48. Stappert, F., Jonsson, J., Mottok, J., Johansson, R.: A Design Framework for End-To-End Timing Constrained Automotive Applications. In: Embedded Real-Time Software and Systems (ERTS), 2010

49. Tindell, K.W.: Using offset information to analyse static priority preemptively scheduled task sets. Tech. Rep. YCS 182, Dept. of Computer Science, University of York (1992)

50. Tindell, K., Clark, J.: Holistic schedulability analysis for distributed hard real-time systems. Microprocess. Microprogram. 40, 117–134 (April 1994)

51. Tindell, K., Hansson, H., Wellings, A.: Analysing real-time communications: controller area network (CAN). In: Real-Time Systems Symposium (RTSS) 1994. pp. 259 –263

Appendix A

Acronyms and Abbreviations

ACC	Autonomous Cruise Control
API	Application Programming Interface
AUTOSAR	AUTomotive Open System ARchitecture
BC	Brake Control
BCRT	Best Case Response Time
CAN	Controller Area Network
CC	Cruise Control
DR	Data Reaction
DRE	Distributed Real-time Embedded
DT	Distributed Transaction
EC	Engine Control
E2EDA	End To End Delay Analysis
FIFO	First In First Out
FILO	First In Last Out
HCAN	Hägglunds Controller Area Network
HRTA	Holistic Response Time Analysis
ICCM	Intermediate Compiled Component Model
ICE	Integrated Component development Environment
ISWC	Input Software Circuit
LIFO	Last In First Out
LILO	Last In Last Out
OSWC	Output Software Circuit
RCM	Rubus Component Model
RTA	Response Time Analysis
SWC	Software Circuit
TIMMO	TIMing MOdel
TP	Timed Path
TTCAN	Time Triggered Controller Area Network
UI	User Interface
WCET	Worst Case Execution Time
WCRT	Worst Case Response Time

Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin

Saad Mubeen is a PhD student at Mälardalen Real-Time Research Centre (MRTC), Mälardalen University, Sweden. His research focus is on modeling and timing analysis of distributed real-time embedded systems in the automotive domain. Saad received his degree of Licentiate in Computer Science and Engineering from Mälardalen University in January 2012. He received his degree of M.Sc. in Electrical Engineering with specialization in Embedded Systems from Jönköping University (Sweden) in 2009. He has co-authored over 30 research papers in peer-reviewed conferences, workshops, books and journals.

Jukka Mäki-Turja is a senior lecturer and researcher at Mälardalen Real-Time Research Centre. His research interest lies in design and analysis of predictable real-time systems. Jukka received his PhD in computer science from Mälardalen University in 2005 with response time analysis for tasks with offsets as focus. He has co-authored over 75 research papers in peer-reviewed conferences, workshops and journals.

Mikael Sjödin is a professor of real-time system and research director for Embedded Systems at Mälardalen University, Sweden. His current research goal is to find methods that will make embedded-software development cheaper, faster and yield software with higher quality. Concurrently, Mikael is also been pursuing research in analysis of real-time systems, where the goal is to find theoretical models for real-time systems that will allow their timing behavior and memory consumption to be calculated. Mikael received his PhD in computer systems in 2000 from Uppsala University (Sweden). Since then he has been working in both academia and in industry with embedded systems, real-time systems, and embedded communications. Previous affiliations include Newline Information, Melody Interactive Solutions and CC Systems. In 2006 he joined the Mälardalen Real-Time Research Centre (MRTC) faculty as a full professor with speciality in real-time systems and vehicular software-systems. He has co-authored over 200 research papers in peer-reviewed conferences, workshops, books and journals.