DOI: 10.2298/CSIS120614011M

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. The safety-critical nature of many real-time 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, 45] 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) [48, 47, 42]

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

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 (except first) receives a trigger, data or both from its predecessor. One way to classify these chains is as trigger and data. In trigger chains, there is only one triggering source (e.g, event, clock or interrupt) that activates the first task. 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.

The end-to-end delays should also be computed along with the holistic response times to predict complete timing behavior of multi-rate real-time systems [21]. For this purpose, the research community has developed the End-to-End[3] 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 these systems should be equipped with the state-of-the-art timing analysis.

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.

Goals and Contributions. In this paper[4], we discuss the implementation of HRTA and E2EDA as two individual plug-ins in the existing industrial tool suite Rubus-ICE (Integrated Component development Environment) [1]. Our goal is to transfer the state-of-the-art real-time analysis results, i.e., HRTA and E2EDA to the existing tools for the industrial use. We discuss and solve the problems encountered, solutions proposed and experiences gained during the implementation, integration and evaluation of the plug-ins. We also provide a proof of concept by conducting the automotive-application case study. These new plug-ins support complete end-to-end timing analysis of DRE systems. Thus, the scope and usability of Rubus tools has widened with the addition of these plug-ins.

[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.

[4] This work is the extension of our previous work [37].

Paper Layout. Section 2 presents the background and related work. Section 3 discusses the end-to-end timing requirements and the implemented analysis. Section 4 describes the challenges encountered, solutions proposed and experiences gained during the implementation and integration of the plug-ins. In Section 5, we present a case study by modeling and analyzing the automotive DRE application. Section 6 concludes the paper and presents the future work.

2. Background and Related Work

2.1. The Rubus Concept

Rubus is a collection of methods and tools for model- and component-based development of dependable embedded real-time systems. Rubus is developed by Arcticus Systems [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) [27] 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 them in terms of data and control flow separately. The control flow is expressed by triggering objects such as internal periodic clocks, interrupts and events. In RCM, the basic component is called Software Circuit (SWC). Its execution semantics are: upon triggering, read data on data *in-ports*; execute the function; write data on data *out-ports*; and activate the output trigger.

RCM separates the control flow from the data flow among SWCs within a node. Thus, explicit synchronization and data access are visible at the modeling level. One important principle in RCM is to separate functional code and infrastructure implementing the execution model. RCM facilitates analysis and reuse of components in different contexts (SWC has no knowledge how it connects to other components). The component model has the possibility to encapsulate SWCs into software assemblies enabling the designer to construct the system at different hierarchical levels. Recently, we extended RCM for the development of DRE systems by introducing new components [30, 39, 33]. A detailed comparison of RCM with several component models is presented in [39].

Fig. 1(a) depicts the sequence of main steps followed in Rubus-ICE from modeling of an application to the generation of code. 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, the coder tool generates the code.

2.2. 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.

2.3. Plug-in Framework in Rubus-ICE

The plug-in framework in Rubus-ICE [26] facilitates the implementation of research results in isolation (without needing Rubus tools) and their integration as add-on plug-ins (binaries or source code) with the Rubus-ICE. A plug-in is interfaced with the builder tool as shown in Fig. 1(a). 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. 1(b) shows the conceptual organization of a plug-in in the Rubus-ICE.

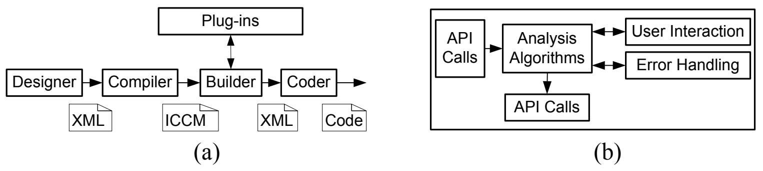


Fig. 1. Sequence of steps from design to code generation in Rubus-ICE

2.4. Response-Time Analysis

RTA of Tasks in a Node. Liu and Layland [28] provided theoretical foundation for analysis of fixed-priority scheduled systems. Joseph and Pandya published the first RTA [25] for the simple task model presented in [28]. Subsequently, it has been applied and extended in a number of ways by the research community. 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 [40]. Tindell [47] developed the schedulability analysis for tasks with offsets for fixed-priority systems. It was extended by Palencia and Gonzalez Harbour [42]. Later, Mäki-Turja and Nolin [29] reduced pessimism from RTA developed in [47, 42] and presented a tighter RTA for tasks with offsets by accurately modeling inter-task interference. We implemented tighter version of RTA of tasks with offsets [29] as part of the HRTA and E2EDA.
RTA of Messages in a Network. To stay focussed on the automotive or vehicular domain, we will consider only Controller Area Network (CAN) and its high-level protocols. Tindell et al. [49] 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 [49, 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 [32, 36, 31], 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. [38] 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. 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. 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 [48]. In [34], 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. [46] 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 [43]. In this work, we will implement the end-to-end delay analysis of [21].

2.5. Tools for End-to-end Timing Analysis of 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 [47, 42] whose tighter version [29] is implemented as part of the HRTA and E2EDA plug-ins. It 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. Volcano Network Architect (VNA) [12] is a communication design tool that supports the analysis of Local Interconnect

Network (LIN) and CAN. It also supports end-to-end timing analysis of a system with more than one network. It implements RTA of CAN presented in [49].

SymTA/S [23] 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 [44].

The Rubus-ICE tool suite allows a developer to specify timing information and perform the HRTA and E2EDA 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 [32], RTA of mixed messages with offsets [38] and a tighter version of offset-based RTA algorithm [29] as part of the HRTA and E2EDA .

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 (single-node or distributed) can be modeled with trigger chains (see Fig.2(a)), data chains (see Fig.2(b)) 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 end-to-end deadline requirements are placed on the holistic response times.

An example of a trigger chain that consists of three components is shown in Fig. 2(a). 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, the end-to-end timing requirements correspond to the holistic response times. Distributed real-time systems can also be modeled with trigger chains in a similar fashion.

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. 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, there may be 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 at the output [21]. 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 has traversed the chain from first to last task) [43]. 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. 2(b). 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 $1ms$. The time line corresponding to the run-time execution of the three tasks (corresponding to three SWCs), depicted in Fig. 3, shows multiple outputs corresponding to a single input. The four end-to-end delays are also identified.

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 and corresponding last response of task τ_C. This delay is identified as "Data Age"[5] in [21]. Data age specifies the

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

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.

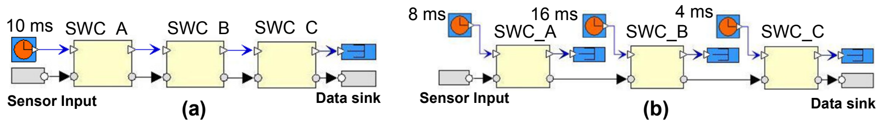


Fig. 2. RCM model of (a) trigger chain (b) data chain in a single-node real-time system

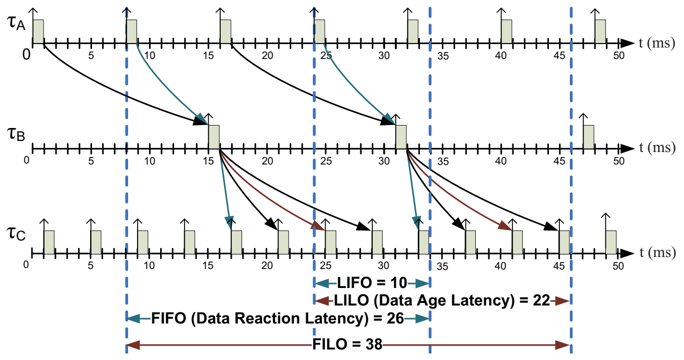


Fig. 3. End-to-end delays for a data chain in a real-time system

First In First Out (FIFO). This delay is equal to the time elapsed between the previous non-overwritten release of task τ_A and first response of task τ_C 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. 3. 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. This delay is identified as "first reaction or Data Reaction"[6] in [21]. It is equal to the longest allowed reaction time for data produced by the initiator to be delivered to the terminator. It finds its importance in button-to-reaction applications in body electronics domain where first reaction to input is important.

[6] We will consistently use the term "Data Reaction delay" to refer to FIFO delay.

First In Last Out (FILO). It is equal to the time elapsed between the previous non-overwritten release of task τ_A and last response of task τ_C corresponding to current non-overwritten release of task τ_A. The reasoning about "just missing" a fresh input (in the case of FIFO delay) is also applicable in this case.

The modeling of data chains and the definition of their end-to-end delays in distributed real-time systems is done in a similar fashion.

3.3. Implemented Holistic Response-Time Analysis

In order to analyze tasks in each node, we implement RTA of tasks with offsets developed by [47, 42] and improved by [29]. 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 [49, 19]; (2) RTA of CAN for mixed messages [32]; (3) RTA of CAN for mixed messages with offsets [38] (The analysis of this profile is implemented as a standalone analyzer).

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 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 plug-in is shown in Fig. 4.

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. 2(b) 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. 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. 4. The pseudocode of E2EDA algorithm (see [21] for details) is shown in Algorithm 2.

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
```

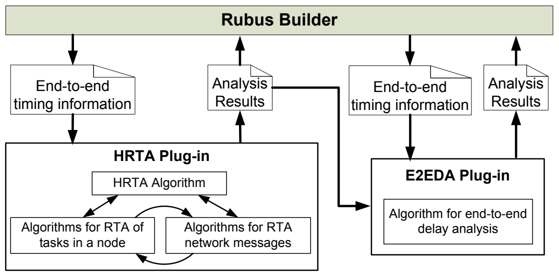

Rubus Builder
End-to-end timing information
Analysis Results
End-to-end timing information
Analysis Results
HRTA Plug-in
HRTA Algorithm
Algorithms for RTA of tasks in a node
Algorithms for RTA network messages
E2EDA Plug-in
Algorithm for end-to-end delay analysis

Fig. 4. Conceptual view of the E2EDA plug-in in Rubus-ICE

4. Encountered Problems, Solutions and 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.

4.1. Extraction of Unambiguous Timing Information

One common assumption in end-to-end response time and delay analyses is that the timing attributes 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

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$^{st}$instance), $\tau_B$(1$^{st}$instance), $\tau_C$(5$^{th}$instance)] in Fig. 3 is a valid TP. On the other hand, TP [$\tau_A$(1$^{st}$instance), $\tau_B$(1$^{st}$instance), $\tau_C$(1$^{st}$instance)] in Fig. 3 is invalid because information cannot flow between $\tau_B$(1$^{st}$instance) and $\tau_C$(1$^{st}$instance)
 4: procedure COMPUTE_FF_DELAY(FF_TP)
 5:     FF_delay = $\alpha_n$(instance) + $\delta_n$(instance) - $\alpha_1$(instance) ▷ $\alpha_n$(instance): Activation time of the corresponding instance of the n$^{th}$ task in timed path FF_TP
            ▷ $\delta_n$(instance): Response time of the corresponding instance of the n$^{th}$ 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$^{nd}$instance), $\tau_B$(1$^{st}$instance), $\tau_C$(5$^{th}$instance)] in Fig. 3
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
```

analysis models are often build upon different meta-models [22]. Moreover, the design model can contain redundant timing information. Hence, it is not trivial to extract unambiguous timing information for HRTA and E2EDA. The timing information (to be extracted) can be divided into two categories.

Extraction of Timing Information Corresponding to User Inputs. The first category corresponds to the timing attributes of tasks 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 [33], we identified all the timing attributes of nodes, networks, transactions, tasks and messages that are required by the HRTA.

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 periodic transmission) or message inhibit time (in 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.

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. 5. It should be noted that the Out Software Circuit (OSWC), shown in the figure, is one of the network interface components in RCM that sends a message to the network. Similarly, In Software Circuit (ISWC) receives a message from the network [39].

In Fig. 5(a), the sender task is activated by a clock, and hence the corresponding message is periodic. Similarly, the corresponding message is sporadic in Fig. 5(b) because the sender task is activated by an event. However, the sender task in Fig. 5(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 in the CANopen protocol [4] and AUTOSAR communication [9] then such message will be treated as a special type of sporadic message. If triggering sources are independent of each other (e.g., in the HCAN protocol [15]), the corresponding message will be considered a mixed message [32, 36]. If there are periodic and sporadic messages in the application, the HRTA plug-in uses the first profile for network analysis (see Section 3.3). On the other hand, if the application contains mixed messages as well, the second profile for network analysis is used.

Identification of Trigger, Data and Mixed Chains. The end-to-end timing requirements on trigger chains are different from those on data chains. These requirements 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 operate. 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 specified on this task in the case of dependent triggering.

However, a system can also be modeled with mixed chains that are comprised of data chains as well as trigger chains as shown in Fig. 5(d). 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".

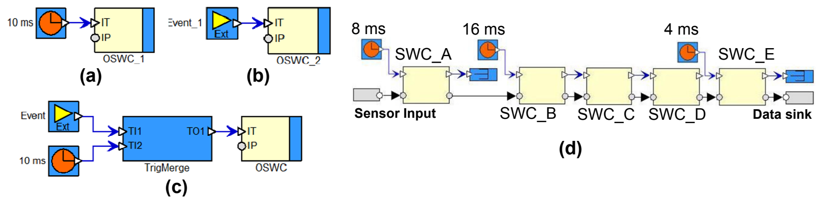


Fig. 5. Extraction of transmission type of a message, (d) RCM model of a mixed chain

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 [35]. Consider the following DT in a two-node DRE system shown in Fig. 6. $SWC1 \rightarrow OSWC\_A \rightarrow ISWC\_B \rightarrow SWC2 \rightarrow SWC3$ We identified the need for the following mappings in the component model: between signals and input data ports of OSWCs at the sender node; between signals and the outgoing message at the sender node; between data output ports of ISWC components and the signals (to be sent to the desired components) at the receiver node; between received message and signals at the receiver node; between multiple signals (structure of signals) and a complex data port; and among all trigger ports of network interface components along a DT.

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 extracted.

4.3. Analysis of Distributed Transactions with Branches

Consider the example of a two-node DRE system containing branches in DTs as shown in Fig. 7. $OSWC\_A1$ and $OSWC\_A2$ in node A send messages $m1$

and $m2$ that are received by $ISWC\_C1$ and $ISWC\_C2$ in node C respectively. Hence, there are two DTs that have different initiators but a single terminator, i.e., SWC_C3 as shown below.
1. $SWC\_A1 \rightarrow SWC\_A2 \rightarrow OSWC\_A1 \rightarrow ISWC\_C1 \rightarrow SWC\_C1 \rightarrow SWC\_C3$
2. $SWC\_A3 \rightarrow OSWC\_A2 \rightarrow ISWC\_C2 \rightarrow SWC\_C2 \rightarrow SWC\_C3$

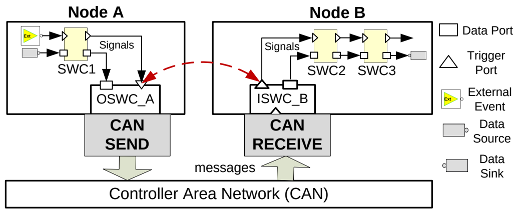


Fig. 6. Two-node DRE system modeled with RCM

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). The calculations for Data Age delay require the response time of SWC_C3. However, the response time of this task depends upon the holistic response times of both DTs. In this case, the HRTA plug-in will calculate the holistic response times of all branches whereas the E2EDA plug-in will consider the maximum value among these holistic response times during calculations for the end-to-end delays.

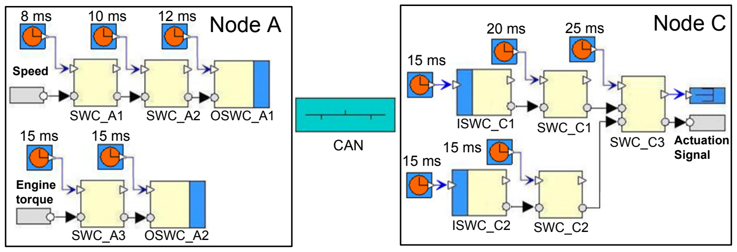


Fig. 7. RCM model of a two-node DRE system with branches in distributed transactions

4.4. Analysis of Mixed Task 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. Using this option, the execution time line of the task chain corresponding to component chain of Fig. 5(d) is shown in Fig. 8(a). This time line will be used by the E2EDA plug-in to calculate the total number of timed paths. However, there are several timed paths, indicated with crosses in Fig. 8(a), 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.

Instead, we use the second option that reduces the number of paths in mixed chain by combining all tasks belonging to a trigger sub-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. 5(d) is shown in Fig. 8(b). The corresponding end-to-end delays are also identified. By implementing the second option , we got rid of the so-called “impossible timed paths”. 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. Path reduction in distributed mixed chains is done in a similar fashion.

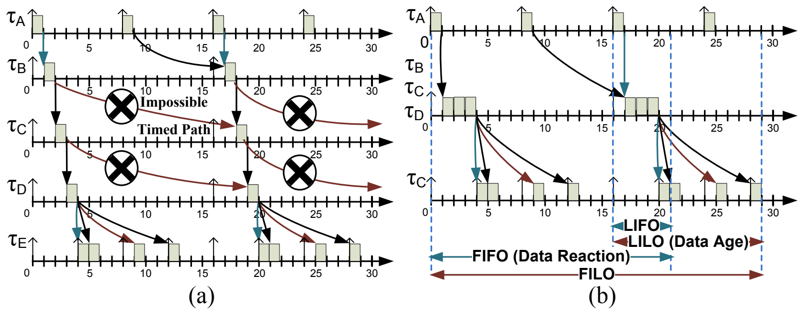


Fig. 8. (a) Impossible timed paths in mixed chains (b) Reduction of a mixed chain

4.5. Analysis of the System Containing “Outside” Messages

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. Hence, 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.

The HRTA connects the tasks and messages in a DT by means of attribute inheritance [48]. Moreover, the message also inherits other attributes from the sender task such as transmission type (periodic, sporadic or mixed [32]); and period or inhibit time or both. 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, each such message is assumed to be the initiator of the corresponding DT. The transmission type and period (or inhibit time or both) of this 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, i.e., the receiver task will inherit the difference between the worst- and best-case response times of the message as its release jitter.

4.6. Impact of Component Technology on the Analysis Implementation

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 technology), 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 component may differ among different component technologies.

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 HRTA may run forever (if deadlines are not specified) because the response times increase in every iteration. Consider a two-node application modeled in RCM as shown in Fig. 9 (a). The $OSWC\_A$ component in node A sends a message $m1$ to node B where it is received by $ISWC\_B$. Similarly, $OSWC\_B$ in node B sends a message $m2$ to $ISWC\_A$ in node A.

There are two options for the run-time allocation of network interface component (OSWC or ISWC) as shown in Fig. 9 (b). First option is to allocate it to the task that corresponds to the immediate SWC, i.e., 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. 9 (b). Similarly, τ_B is the run-time representation of $ISWC\_B$, SWC_B and $OSWC\_B$. Obviously, this run-time allocation will result in direct cycles. This problem may appear in those component technologies 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 technologies.

The direct cycles can be avoided by allocating each network interface component to a separate task as shown in the option 2 in Fig. 9 (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. Hence, the HRTA may produce converging results, 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.

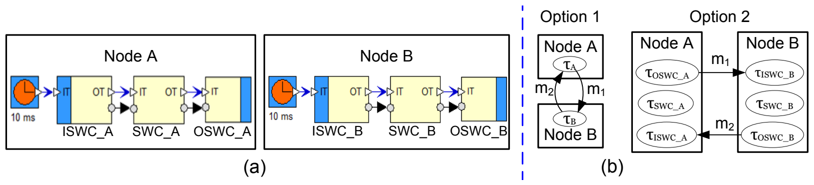


Fig. 9. Options for the run-time allocation of network interface components

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. 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 as shown in Fig. 10. The option A supports reusability by building the HRTA plug-in by integrating existing RTA plug-in with two new plug-ins, i.e., one implementing network RTA and the other implementing the HRTA. In this case, the 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 the HRTA. This option does not support any reuse of existing plug-ins.

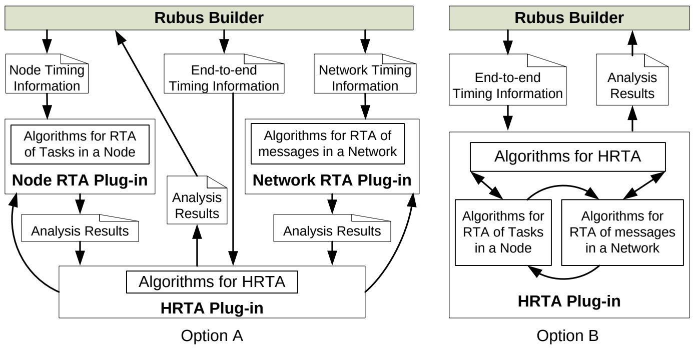

Rubus Builder
Node Timing Information
End-to-end Timing Information
Network Timing Information
Algorithms for RTA of Tasks in a Node
Node RTA Plug-in
Analysis Results
Algorithms for RTA of messages in a Network
Network RTA Plug-in
Analysis Results
Analysis Results
Algorithms for HRTA
HRTA Plug-in
Option A
Rubus Builder
End-to-end Timing Information
Analysis Results
Algorithms for HRTA
Algorithms for RTA of Tasks in a Node
Algorithms for RTA of messages in a Network
HRTA Plug-in
Option B

Fig. 10. Options to develop the HRTA Plug-in for Rubus-ICE

Since, option A allows the reuse of a pre-tested and heavyweight 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. Hence, we selected option B for the implementation of the HRTA. Since E2EDA algorithm is non-iterative, 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. 4.

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. Such transaction is divided into sub- transactions (each having a single network) which are analyzed separately in the first step. In the second step, the attribute inheritance is carried out (see Section 3.3) and the sub-transactions are analyzed again. The second step is repeated until the response times converge or the deadlines (if specified) are violated. Although, we analyze the sub-transactions 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 times from the HRTA plug-in, it does not need to split the system into sub-systems.

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 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. 11 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. A delay constraint can also be distributed over several nodes. Another useful method for specifying the delay constraints is by selecting each component (e.g., with mouse click) along the 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 times of all tasks, messages and DTs in the system because it may contain a lot of useless information (if the user is not interested in all of it). 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. 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 [32].

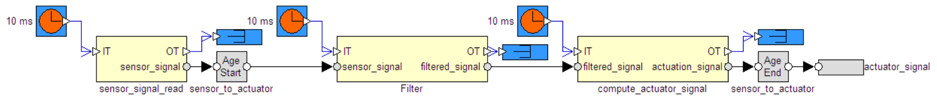


Fig. 11. Age delay constraint specified on a data path

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

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. Further, the user should be able to interact with the plug-in, i.e., 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. 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 support for this type of feedback in the HRTA plug-in will be provided in the future.

4.14. Continuous Collaboration between Integrator and Implementer

Our experience 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. 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. Automotive Application Case Study

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

5.1. Autonomous Cruise Control System

A Cruise Control (CC) 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 CC 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 [41]. The ACC system may be divided into four subsystems, i.e., Cruise Control (CC), Engine Control (EC), Brake Control (BC) and User Interface (UI) [14] as shown in Fig. 12. The subsystems communicate with each other via the CAN network.

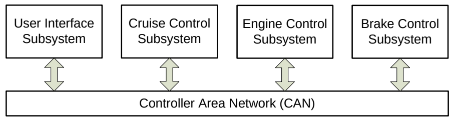


Fig. 12. Block diagram of Autonomous Cruise Control System

User Interface (UI) Subsystem. It 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. It also receives messages that include linear and angular speed of the vehicle, status of manual brake sensor, state of ACC subsystem, status messages and warnings to be displayed on the screen. It also sends messages (including status of driver's input) to other subsystems.

Cruise Control (CC) Subsystem. The CC subsystem receives user input information as a CAN message from the UI subsystem. From the received message it analyzes the state of the CC switch; if it is in ON state then it activates the CC 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 BC and EC 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 BC subsystem. If brakes are pressed manually then the CC functionality is disabled. It also sends status messages to the UI subsystem.

Engine Control (EC) Subsystem. The 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 from other subsystems that include information regarding vehicle speed, status of manual brake sensor, and input information processed by the UI system. Based on this 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 (BC) Subsystem. The 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 status of manual brake, vehicle speed and RPM.

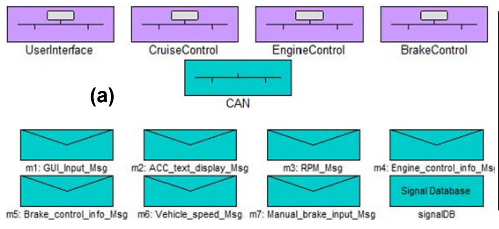
5.2. Modeling of ACC System with RCM in Rubus-ICE

In RCM, we model each subsystem as a separate node connected to a CAN network as shown in Fig. 13(a). The selected speed of the CAN bus is 500 kbps. The extended frame format is selected, i.e., each frame will use 29-bit identifier [24]. The ACC system is modeled with trigger, data and mixed chains.

There are seven CAN messages in the system as shown in Fig. 13(b). A signal data base "signalDB" that contains all the signals sent to the network is also shown. Each signal in the signalDB 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 inhibit time (T_m) are listed in the table shown in Fig. 13(c). The high-level architectures of CC, EC, BC and UI nodes modeled with RCM are shown in Fig. 14(a), 14(b), 14(c) and 14(d) respectively.

Internal Model of CC Node in RCM. The CC node is modeled with four assemblies as shown in Fig. 14(a). 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. 15. 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. 16(a). These components receive messages $m1$, $m6$ and $m7$ from the CAN respectively. The assembly Output_to_CAN contains three OSWC components that send messages $m5$,

$m4$ and $m2$ to the CAN network as shown in Fig. 16(b). The Cruise_Control assembly contains two SWCs: one handles the input and CC 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. 17.



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

(c)

Fig. 13. (a) RCM model of ACC system, (b) RCM model of CAN messages and signal database, (c) message attributes extracted from the model

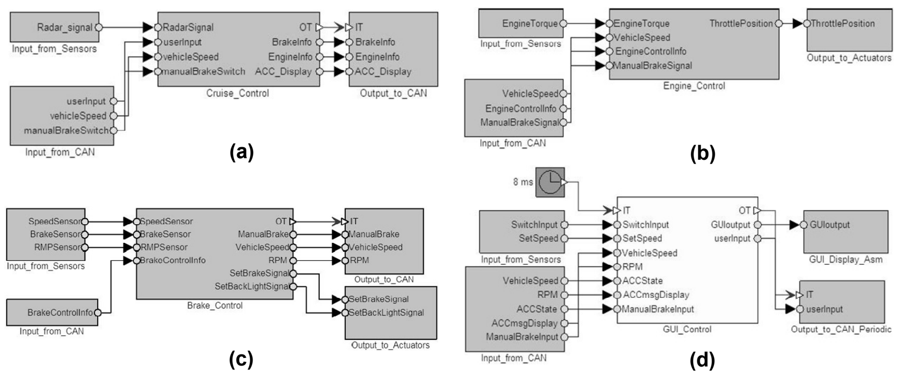


Fig. 14. RCM model of (a) CC node, (b) EC node, (c) BC node, (d) UI node

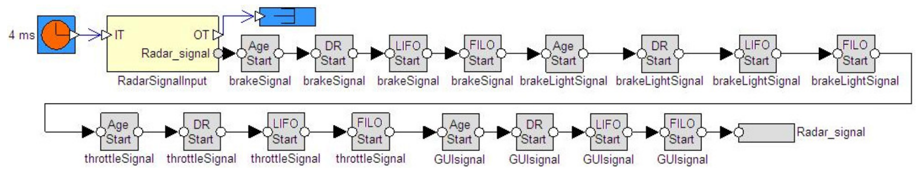


Fig. 15. CC node: Internal model of the Input_from_Sensors assembly

Internal Model of EC Node in RCM. The EC node is modeled with four assemblies as shown in Fig. 14(b). The Input_from_Sensors assembly contains one SWC that reads the sensor values corresponding to the engine torque as shown in Fig. 18(a). 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. 18(b). These components receive messages $m6$, $m4$ and $m7$ from the CAN network respectively. The third assembly, Output_to_Actuators, shown in Fig. 18(c), contains the SWC that produces control signals for the engine throttle actuator. The fourth assembly Engine_Control, shown in Fig. 19, 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 of a distributed mixed chain that we will analyze along with other distributed mixed chains in the next subsections.

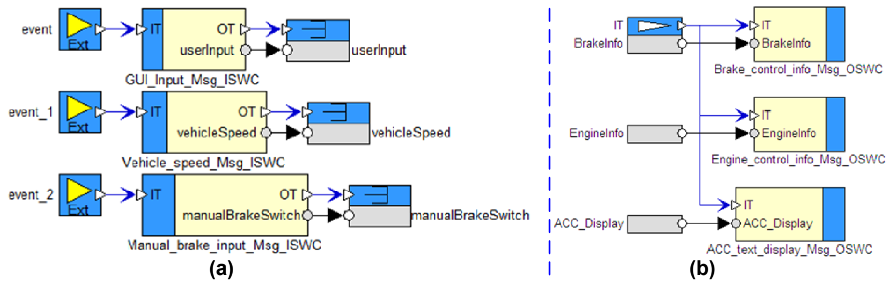


Fig. 16. CC node: Internal model of assemblies (a) Input_from_CAN, (b) Output_to_CAN

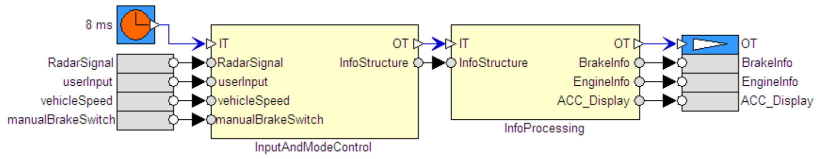


Fig. 17. CC node: SWCs comprising the Cruise_Control assembly

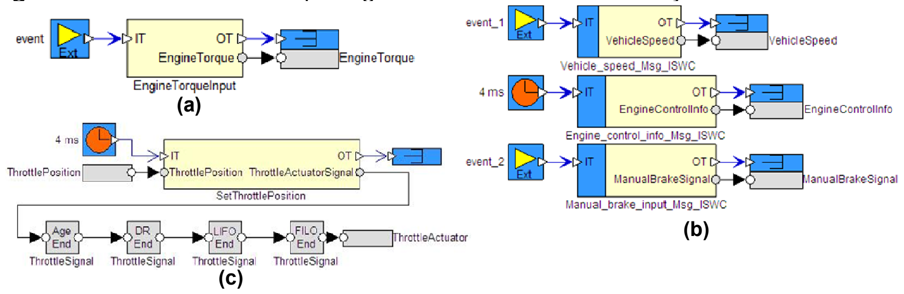


Fig. 18. EC node: Internal model of assemblies (a) Input_from_Sensors, (b) Input_from_CAN, (c) Output_to_Actuators

Internal Model of BC Node in RCM. The BC node is modeled with five assemblies as shown in Fig. 14(c). The Input_from_Sensors assembly contains three SWCs as shown in Fig. 20(a). 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. 20(b), contains the ISWC component Brake_control_info_Msg_ISWC that receives a message $m5$ from the CAN. The third assembly, i.e., Brake_Control as shown in Fig. 21(a), 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. 20(c). These components send messages $m7$, $m6$ and $m3$ to the CAN. The fifth assembly, Output_to_Actuators as shown in Fig. 21(b), contains the SWCs that produce control signals for the brake actuators and brake light controllers.

Internal Model of UI Node in RCM. The UI node is modeled with four assemblies along with one SWC as shown in Fig. 14(d). The GUI_Control SWC handles the input from the sensors and messages from the CAN. 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. 22(a). 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. 22(b). These components receive messages $m6$, $m3$, $m7$ and $m2$ from the CAN respectively. The third assembly, i.e., Output_to_ CAN_Periodic sends a message $m1$ to the CAN via the OSWC component as shown in Fig. 22(c). The fourth assembly, i.e., GUI_Display_Asm contains one SWC, i.e., GUIdisplay component as shown in Fig. 23. This component sends the signals (corresponding to updated information) to GUI in the car.

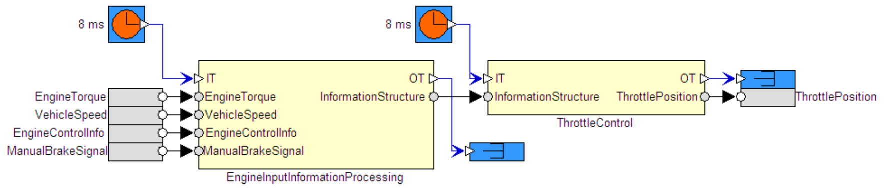


Fig. 19. EC node: SWCs comprising the Engine_Control assembly

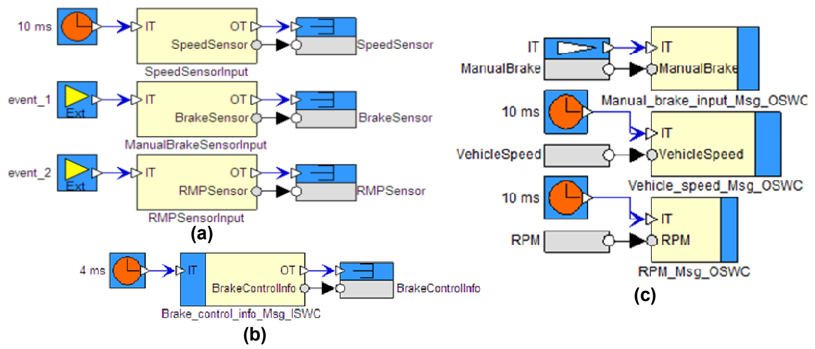


Fig. 20. BC node: Internal model of assemblies (a) Input_from_Sensors, (b) Input_from_CAN, (c) Output_to_CAN

5.3. Modeling of End-to-end Deadline Requirements

We specify end-to-end deadline requirements on four DTs in the ACC system using the 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 1. 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 CC node as shown in Fig. 15. The last tasks of DT$_1$ and DT$_2$ are located in the BC node. These tasks correspond to the SWCs *SetBrakeSignal* and *SetBrakeLightSignal* as shown in Fig. 14(c). 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 EC node as shown in Fig. 14(b). It produces control signal for the engine throttle actuator. The last task of DT$_4$ corresponds to *GUIdisplay* SWC and is located in the UI node as shown in Fig. 14(d). This task provides display information for the driver.

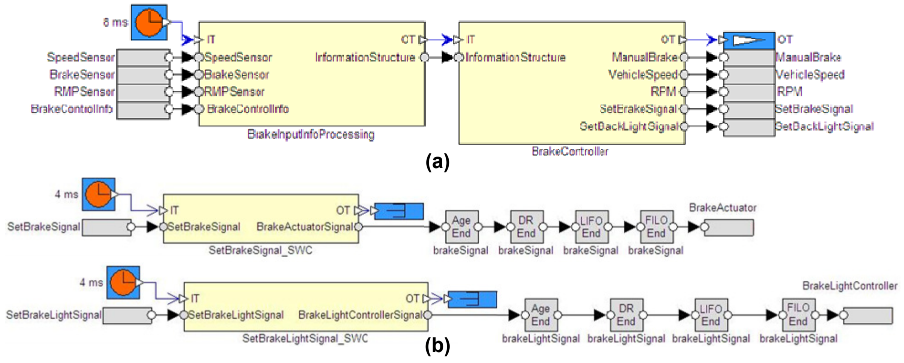


Fig. 21. BC node: Model of assemblies (a) Brake_Control (b) Output_to_Actuators

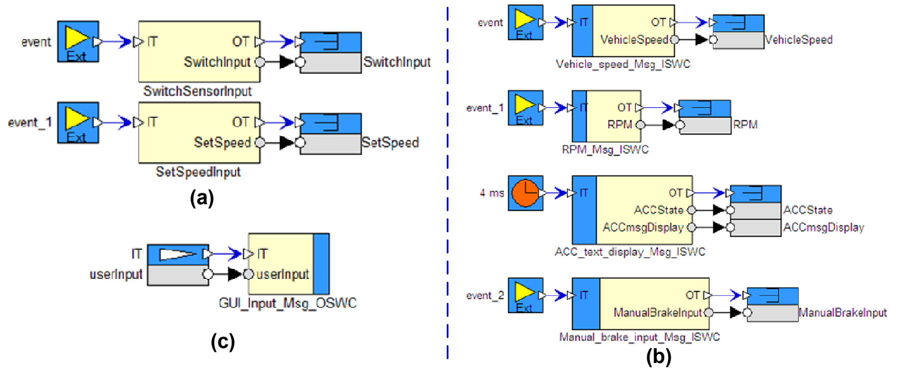


Fig. 22. UI node: Internal model of assemblies (a) Input_from_Sensors, (b) Input_from_CAN, (c) Output_to_CAN_Periodic

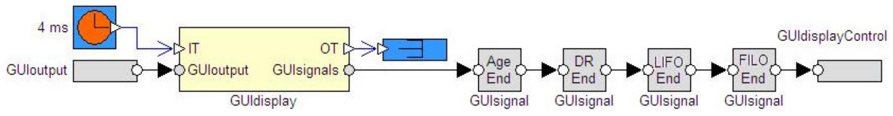


Fig. 23. UI node: Internal model of the GUI_Display_Asm assembly

All the mixed chains under analysis are distributed over more than one node. 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. 15. There are sixteen start objects for delay constraints in Fig. 15 because there are four DTs under analysis with four delay constraints specified on each. The end objects for all delay constraints for DT$_1$ and DT$_2$ are specified in Fig. 21(b). Similarly, the end objects for all delay constraints for DT$_3$ and DT$_4$ are specified in Fig. 18(c) and Fig. 23 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$_4$: $RadarSignalInput \rightarrow InputAndModeControl \rightarrow InfoProcessing \rightarrow ACC\_text\_display\_Msg\_OSWC \rightarrow message : m2 \rightarrow ACC\_text\_display\_Msg\_ISWC \rightarrow GUI\_Control \rightarrow GUIdisplay$

5.4. 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 listed in the table shown in Fig. 13(c). 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 1 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 GUI. The analysis report produced by the E2EDA plug-in is shown in Table 2. It lists four end-to-end delays calculated for each DT. 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 1 and 2 respectively, we see that the modeled ACC system meets all of its deadlines.

6. 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 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.

Table 1. 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

Table 2. 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

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 plug-ins. The experience gained by dealing with the implementation challenges provided a feed back to the component technology. 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 ACC system with component-based 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. 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 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 with FIFO and work-conserving queues [18, 20], and RTA of CAN with FIFO Queues for Mixed Messages [36] within HRTA plug-in. We also plan to integrate the stand alone analyzer, that we developed for the analysis of mixed messages with offsets [38], with the HRTA plug-in.

References

1. Arcticus Systems, http://www.arcticus-systems.com
2. BAE Systems Hägglunds, http://www.baesystems.com/hagglunds
3. CANoe. 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 RT Applications, http://mast.unican.es
7. Mecel, web page, http://www.mecel.se
8. RAPID RMA: The Art of Modeling Real-Time Systems, 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
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
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 (Oct 2009)
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. pp. 33 –42 (May 2012)
19. Davis, R., Burns, A., Bril, R., Lukkien, J.: Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised. RTS 35, 239–272 (2007)
20. Davis, R.I., Kollmann, S., Pollex, V., Slomka, F.: Controller Area Network (CAN) Schedulability Analysis with FIFO queues. In: ECRTS 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: CRTS, 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 (Oct 2010)
23. Hamann, A., Henia, R., Racu, R., Jersak, M., Richter, K., Ernst, R.: Symta/s - symbolic timing analysis for systems (2004)
24. ISO 11898-1: Road Vehicles interchange of digital information controller area network (CAN) for high-speed communication, ISO Standard-11898, Nov. 1993.
25. Joseph, M., Pandya, P.: Finding Response Times in a Real-Time System. The Computer Journal (British Computer Society) 29(5), 390–395 (October 1986)

26. K. Hänninen et.al.: Framework for real-time analysis in Rubus-ICE. In: 13th IEEE Conference on Emerging Technologies and Factory Automation) (2008)
27. K. Hänninen et.al.: The Rubus Component Model for Resource Constrained Real-Time Systems. In: 3rd IEEE Symposium on Industrial Embedded Systems (2008)
28. Liu, C., Layland, J.: Scheduling algorithms for multi-programming in a hard-real-time environment. ACM 20(1), 46–61 (1973)
29. Mäki-Turja, J., , Nolin, M.: Tighter response-times for tasks with offsets. In: Real-time and Embedded Computing Systems and Applications Conference (August 2004)
30. 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
31. 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 (September 2011)
32. 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)
33. 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)
34. 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, WIP Session. pp. 9–12 (December 2011)
35. 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
36. 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)
37. 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)
38. 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)
39. 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)
40. Nolin, M., Mäki-Turja, J., Hänninen, K.: Achieving Industrial Strength Timing Predictions of Embedded System Behavior. In: ESA. pp. 173–178 (2008)
41. P. Berggren: Autonomous Cruise Control for Chalmers Vehicle Simulator. Master's thesis, Dept. of Signals and Systems, Chalmers University of Technology (2008)
42. Palencia, J., Harbour, M.G.: Schedulability Analysis for Tasks with Static and Dynamic Offsets. IEEE International Symposium on Real-Time Systems p. 26 (1998)
43. 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: EMSOFT, 2010. pp. 129–138. ACM

44. Schmidt, D., Kuhns, F.: An overview of the Real-Time CORBA specification. Computer 33(6), 56 –63 (June 2000)
45. 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)
46. Stappert, F., Jonsson, J., Mottok, J., Johansson, R.: A Design Framework for End-To-End Timing Constrained Automotive Applications. In: ERTS, 2010
47. Tindell, K.W.: Using offset information to analyse static priority preemptively scheduled task sets. Tech. Rep. YCS 182, University of York (1992)
48. Tindell, K., Clark, J.: Holistic schedulability analysis for distributed hard real-time systems. Microprocess. Microprogram. 40, 117–134 (April 1994)
49. Tindell, K., Hansson, H., Wellings, A.: Analysing real-time communications: controller area network (CAN). In: Real-Time Systems Symposium, 1994. pp. 259 –263

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 MRTC. 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 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.

Received: June 14, 2012; Accepted: November 12, 2012.