End-to-end Timing Analysis of Black-box Models in Legacy Vehicular Distributed Embedded Systems

Saad Mubeen, Mikael Sjödin,
Thomas Nolte
Mälardalen Real-Time Research Centre (MRTC),
Mälardalen University, Västerås, Sweden
{saad.mubeen, mikael.sjodin, thomas.nolte}@mdh.se

John Lundbäck, Mattias Gålnander,
Kurt-Lennart Lundbäck
Arcticus Systems AB, Järfälla, Sweden
{john.lundback, mattias.galnander, kurt.lundback}
@arcticus-systems.com

***Abstract*—A majority of existing techniques and tools, used in the vehicular industry, support the extraction of end-to-end timing models. Such models are used to perform timing analysis of distributed embedded systems at an abstraction level that is close to their implementation. This paper takes a first initiative to provide such a support at a higher level of abstraction. At such a level, the system can be modeled with inter-connected black-box models of nodes whose internal software architectures may not be available. However, most of the design decisions about network communication are available. This represents a typical scenario in the vehicular industry where most of the artifacts are reused from either legacy systems, other projects or previous releases of the vehicle. In this paper we present an approach for the extraction of end-to-end timing models at the highest level of abstraction used in the vehicular domain. Using these models, end-to-end path delay analysis of the systems can be performed at a higher abstraction level and at an early phase during the development. As a proof of concept we implement this technique in an industrial tool suite, Rubus-ICE, that is used for the development of these systems by several international companies. Using the extended tool, we conduct a vehicular-application case study.**

I. Introduction

The amount of computer controlled functionality in the vehicular domain has significantly increased over the past few years. As a result, software in vehicular embedded systems has also drastically increased in size and complexity. For example, the embedded software in modern heavy trucks may consist of as many as 2000 software functions which may be distributed over 45 ECUs (Electronic Control Units) [1]. In order to deal with such complexity, the model- and component-based development approach has emerged as an attractive option for these systems [2], [3]. This approach employs the principles of Model Based Software Engineering (MBSE) and Component Based Software Engineering (CBSE). It uses models to describe functions, structures and other design artifacts. It raises the *level of abstraction* for software development by reuse and integration of software components and their architectures. An abstraction level provides a complete definition of the system for a given purpose during the development process. Within the segment of construction-equipment vehicles and similar segments for heavy special-purpose vehicles, model-based development of software architectures for embedded systems has had a surge the last few years [4], [5], [6], [7].

Most of the vehicular distributed embedded systems have real-time requirements. This means, the time at which these systems respond to some stimulus is equally important as logical correctness of the response. Hence, logically correct but late response may be considered as bad as logically incorrect response. The providers of such systems must ensure that the actions by the systems are taken at a time that is appropriate for their environment. One way to guarantee this is to perform end-to-end timing analysis of the system [8], [9]. It can validate timing requirements, specified on the system, without performing exhaustive testing. In order to perform the timing analysis, the end-to-end timing model should be extracted from the component-based software architecture of the system.

A. Objectives and Problem Statement

The majority of existing model- and component-based development approaches, methodologies and analysis tools that are used in the vehicular domain support the extraction of end-to-end timing models at an abstraction level that is close to system implementation. As a result, the end-to-end timing analysis can be performed only at the implementation level. This, in turn, hampers the reuse and refinement of models from legacy systems (previously developed) with respect to timing requirements at higher abstraction levels and earlier phases during the development. Moreover, timing behavior of the system cannot be verified at higher abstraction levels. There are some initiatives that aim to provide end-to-end timing analysis at one abstraction level above the implementation [10], [11], [7]. However, these are ongoing works.

None of the existing modeling techniques and tools that are used in the vehicular industry for the development of legacy distributed embedded systems support end-to-end timing analysis at the highest abstraction level (also known as the vehicle level) [12], [5]. At this level, the internal software architecture of the nodes may not be available. Intuitively, it may not be possible to extract the end-to-end timing model to perform the timing analysis. This brings up a question: *why are we interested in extracting the end-to-end timing model and performing the end-to-end timing analysis at the vehicle level?* The motivation for this activity comes form the development processes for distributed embedded systems that are used in the vehicle industry, especially in the segment of construction equipment and heavy vehicle architectures.

In the industry, most often, the bottom-up development approach is used as a lot of information, models, artifacts and solutions are reused from other projects and legacy systems. According to an estimate, up to 90% of the software can be

reused from other projects or previous releases of the vehicle if model- and component-based software development is used [1]. The infrastructure and platform (e.g., machine, types of ECUs, networks) for the system to be developed are already known. The traditional process for the development of these systems in the industry starts with designing the bus/network communication. At the early stage of the development, usually the focus is on finding answers to the following questions. How many busses will there be in the system? Which nodes will be connected to which bus? How many messages will there be in the system? Which messages will be sent by each node? After finding answers to these questions, the focus is shifted towards the development of functions.

Our vision is that the end-to-end timing model can be extracted and corresponding end-to-end timing analysis can be performed on the models of vehicular distributed embedded systems at the highest abstraction level if design decisions about network communication are already made. However, the precision of the analysis depends upon the level of details about the internal software component architecture of the nodes. Based on such information, we classify the models of nodes into three categories as follows.

1) *White-box models of nodes:* If the internal software component architecture of nodes is completely reused from other projects, earlier releases or previous iterations (during the development), then the end-to-end response-time and delay analyses [9] can be performed with a high precision. This type of analysis support can be easily provided by applying slight modifications to the existing analysis engines, e.g. [13], [14], that already support the end-to-end timing analysis at the implementation level where a complete software architecture is available.
2) *Black-box models of nodes:* If only crude models of nodes are available, the end-to-end delay analysis can be performed with a low precision. This may correspond to earlier stages during the development where the internal software component architecture of nodes is not available. It may also correspond to the case where the complete models of nodes are provided by different tier-1 suppliers and these nodes may be available at a later stage during the development. Whatever may be the cause of unavailability of the internal software architecture of nodes, the design decisions about network communication must be available for the timing analysis to work. In order to support the analysis of such models, a technique is needed to extract the end-to-end timing models.
3) *Gray-box models of nodes:* If partial software architectures of nodes are available then relatively less precise end-to-end response-time and delay analyses can be performed. This could be the case when extended functionality is required compared to a previous release of the vehicle. The timing model extraction technique in the case of black-box models of nodes should be general and robust enough to support the gray-box node models.

In the second and third categories, because the complete timing information may not be available at the highest abstraction level, the end-to-end timing analysis results may not represent accurate timing behavior of the final system. However, these results can provide useful hints on setting and refining the timing requirements in the next iterations and at lower abstraction levels. In the recent few years, one of the focuses of several large EU research projects, that involve both academia and industry, has been on supporting the timing analysis at various abstraction levels and parts of the development process [5], [6], [7]. To the best of our knowledge, none of the existing techniques and analysis tools support extraction of timing models and corresponding timing analysis at the vehicle level in the second and third categories discussed above. In this paper, we take the first step to provide such a support during the model- and component-based development of distributed embedded systems at the vehicle level of abstraction.

B. Paper Layout

The rest of the paper is organized as follows. In Section II, we discuss background and related work. Section III revisits the black-box modeling technique. Section IV discusses the proposed timing model extraction approach. In Section V, we discuss the adaptation of existing timing analysis which is now applicable to the black-box models of nodes. Section VI presents the proof-of-concept implementation and a case study. Finally, Section VII concludes the paper.

II. Background and Related Work

There are several frameworks that can be used for software development of embedded systems. However, the focus in the vehicle industry today is on EAST-ADL [12] and AUTOSAR [4]; the Rubus Component Model (RCM) [15] is also being used. In this work, we focus only on the vehicular domain.

A. Abstraction Levels Considered by Various Methodologies

There are several software development methodologies and languages for vehicular embedded systems [12], [16], [4], [17] that mainly describe four abstraction levels as shown in Fig. 1.

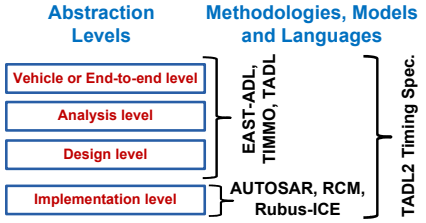


Fig. 1. Abstraction levels considered during the development.

The highest abstraction level, also known as the vehicle or end-to-end level, captures the features and requirements on end-to-end functionality of the vehicle. The information captured at this level is informal and independent of the solution. The requirements are formally captured at the analysis level. The information at this level is independent of the allocation of functions to software/hardware platforms. Further, a high-level analysis may also be performed for functional verification. The artifacts at the design level are developed independent of implementation details. These artifacts also contain middleware abstraction, hardware architecture and software functions to hardware allocation. The implementation level contains a software-based implementation of the system functionality. Basically, it contains the software component architecture of the system. The EAST-ADL methodology defines a system at this level in terms of the AUTOSAR software architecture. Within the vehicle industry, especially in the segment of construction equipment vehicles, RCM is also used at the

implementation level complementary to EAST-ADL at the top three levels.

B. The Rubus Concept

Rubus [18] is a collection of methods and tools for model- and component-based development of embedded real-time systems. It is developed by Arcticus Systems[1] in close collaboration with several industrial partners. It has been in the industrial use for over 20 years. Today, Rubus is mainly used for the development of control functionality in vehicles by several international companies, e.g., BAE Systems, Volvo Construction Equipment, Knorr-bremse, Mecel and Hoerbiger. The Rubus concept is based around RCM and its development environment Rubus-ICE [19] which includes modeling tools, code generators, analysis tools and run-time infrastructure. The main goal of Rubus is to be aggressively resource efficient and to provide means for developing predictable, timing analyzable and synthesizable control functions in resource-constrained embedded systems. The timing analysis supported by Rubus-ICE includes distributed end-to-end response-time and delay analyses [9]. Rubus methods and tools mostly focus at the implementation level in Fig. 1. The lowest-level hierarchical component in RCM is called Software Circuit (SWC) which is used to encapsulate basic functions. Fig. 2 shows an example of a software architecture in RCM composed of SWCs; interconnections between them; and their interactions with external events and actuators. It can be seen in Fig. 2 that RCM clearly separates data flow from control (or trigger) flow.

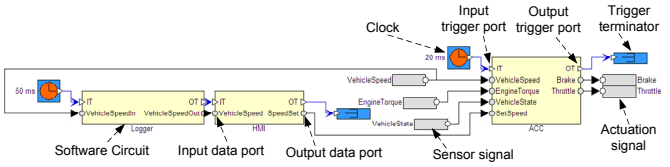


Fig. 2. Example of the software architecture of a system modeled with RCM.

C. AUTOSAR

AUTOSAR is an industrial initiative to support development of a standardized software architecture for automotive embedded systems. It is used at the implementation level. It describes software development at a higher level of abstraction compared to RCM. The timing model in AUTOSAR is introduced fairly recently compared to RCM. AUTOSAR is more focused on the functional and structural abstractions, hiding implementation details about execution and communication. AUTOSAR hides the details that RCM highlights. The timing information required to perform the end-to-end timing analysis can only be extracted at the implementation level in Fig. 1.

D. Related Initiatives and Approaches

TIMMO [5] is an industrial initiative to provide AUTOSAR with a timing model. It is based around a methodology and a language called the Timing Augmented Description Language (TADL) [17] which expresses timing requirements and constraints. It is inspired by MARTE [20] which is a UML profile for model-driven development of real-time and embedded systems. According to the methodology, the EAST-ADL language is used for structural modeling; whereas, AUTOSAR is used at the implementation level. Both TIMMO and EAST-ADL focus at the top three levels in Fig. 1. TADL is redefined and released in the TADL2 specification [16] of the TIMMO-2-USE project [6], [21]. TADL2 is able to specify timing information at all abstraction levels as shown in Fig. 1. Most of these initiatives lack the support for expressing low-level details at the higher levels such as linking information in distributed chains. These details are necessary to extract the end-to-end timing information from the software architecture. Furthermore, there is no support on how to extract this information from the model or how to perform timing analysis. In our view, the end-to-end timing information includes enough information from the systems to be able to perform end-to-end timing analysis. There are several other related model- and component-based development approaches such as COMDES-II [22], ProCom [23] and TECS [24]. To the best of our knowledge, all these languages, methodologies and accompanying tools do not support end-to-end timing analysis at the higher levels of abstraction.

[1] http://www.arcticus-systems.com

E. End-to-end Timing Model

An end-to-end timing model consists of timing properties, requirements, dependencies and linking information of all tasks, messages and task chains in the distributed embedded system under analysis. It can be divided into system timing and linking models. For instance, consider a task chain distributed over three nodes connected by a network as shown in Fig. 3. The system timing model contains all the timing information about the three nodes and the network. Whereas the system linking model contains all the linking information in the task chains, including control and data paths. A method for extracting end-to-end timing models is discussed in [25]. In [10], the translation from the design- to the implementation-level models is provided. However, the translation is done manually. Moreover, the translation is limited by the constraint that it only considers that implementation-level model which results in worst-case response times and delays.

F. End-to-end Timing Analysis

The end-to-end response time of a task chain is defined as the amount of time elapsed between the arrival of an event at the first task and production of the response by the last task in the chain. Consider the end-to-end response time of a task chain as shown in Fig. 3. In this example, arrival of an event corresponds to the brake pedal sensor input in the sensor node, whereas, production of the response corresponds to the brake actuation signal in the actuation node. However, if tasks within a chain are triggered by independent sources (e.g., clocks) then different types of path delays such as *age* and *reaction* should also be calculated. The age delay corresponds to freshness of data and is important in the control systems domain. Whereas, the reaction delay corresponds to the first reaction for a given stimulus and is crucial in button-to-action applications in the body electronics domain.

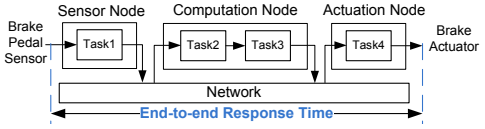


Fig. 3. Example demonstrating end-to-end response time.

In order to explain the meaning of the age and reaction delays, consider a task chain consisting of two tasks τ_1 and τ_2 which are triggered by independent clocks of periods 25ms and 5ms respectively as shown in Fig. 4. Let the first and second instances of τ_1 be denoted by $\tau_1(1)$ and $\tau_1(2)$ respectively. Let the Worst Case Execution Time (WCET) of τ_1 be 2 ms. When τ_1 is triggered it reads data from register Reg-1, performs its computation and then writes data to Reg-2. Similarly upon triggering, τ_2 with WCET of 1 ms reads data from Reg-2, carries out its computation and finally writes data to Reg-3. Since, the tasks are activated independently with different clocks, there can be several paths through which the data can traverse from input (Reg-1) to output (Reg-3) of the chain. We call these data paths as timed paths (a concept borrowed from [26]). These timed paths are shown by several uni-directional arrows in Fig. 5. An example of one of the valid timed paths is $\tau_1(1) \rightarrow \tau_2(2)$, i.e., a data path represented by the first and second instances of τ_1 and τ_2 in Fig. 5. On the other hand, $\tau_1(1) \rightarrow \tau_2(1)$ in Fig. 5 represents an invalid timed path because the data cannot traverse through it from input to output. The age delay is equal to the time elapsed between the current non-overwritten release of τ_1 and corresponding last response of τ_2 among all valid timed paths. Whereas, the reaction delay is equal to the time elapsed between the previous non-overwritten release of τ_1 and the first response of τ_2 corresponding to the current non-overwritten release of τ_1. The age and reaction delays are identified in Fig. 5.

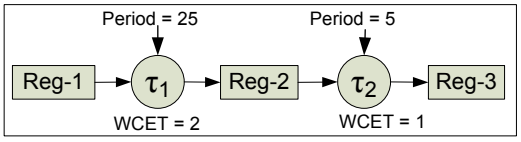


Fig. 4. Example of a task chain with independent activations of tasks.

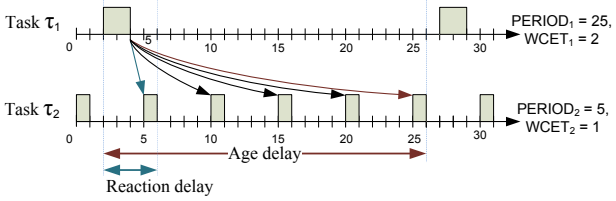


Fig. 5. Example demonstrating end-to-end path delays.

G. Tools Supporting End-to-end Timing Analysis

SymTA/S [13] is a tool by Symtavision for model-based timing analysis and optimization. Among other analyses, it supports end-to-end timing analysis in both single-node and distributed real-time systems. pyCPA [27] is a compositional performance analysis tool that supports the end-to-end timing analysis. CANalyzer [28] supports the simulation, analysis and data logging for systems that use CAN for network communication. The MAST tool suite [29] supports worst-case response-time analysis of distributed systems. It also provides modeling support by integrating itself with a model-based tool chain that is compliant with the MARTE standard. Rubus-ICE is a commercial tool suite that supports end-to-end response time and delay analyses [9]. To the best of our knowledge, all timing analysis tools used in the vehicular domain support end-to-end timing analysis at the implementation level. There is an ongoing work to support the end-to-end timing analysis at the design level in Rubus-ICE [11], [7]. In comparison, this paper provides a support to perform end-to-end timing analysis at the vehicle level. As a proof of concept, we implement the corresponding timing models extraction method and the end-to-end timing analysis in Rubus-ICE.

H. Relation with the Authors' Previous Works

In [30], we presented a method to extract end-to-end timing models from distributed embedded systems at the implementation level. Using the extracted models, we provided analysis engines to perform end-to-end timing analysis [9]. In [10], [11], we explored the timing model extractions using model transformations at the design level. As a proof of concept, we extracted timing information from the systems developed with EAST-ADL using the TIMMO methodology; and annotated with timing information using TADL2. At the implementation level, the method exploits RCM and Rubus-ICE to extract the timing information that cannot be clearly specified at the design level. In [31], we proposed an approach to support modeling of legacy distributed embedded systems at the vehicle level. However, the approach does not support the extraction of end-to-end timing models. In comparison, our current work provides an approach to extract and support end-to-end timing models and analysis respectively at the vehicle level. It is a step towards the development of a seamless tool-chain for model-based development of vehicular embedded systems; and support for inter-operation of various modeling and analysis tools, including the AUTOSAR-based tool chain [4], [7].

III. Revisiting the Black-box Modeling Approach

There are a number of techniques that are used for modeling of vehicular distributed embedded systems at the vehicle abstraction level such as [12], [6]. However, a majority of these techniques have limited support for modeling legacy systems. In this section, we briefly revisit the black-box modeling approach [31] that provides a comprehensive support to model not only crude or black-box nodes but also legacy nodes whose internal software architectures are available for reuse.

This approach proposes to use a black-box node model to represent the node whose internal software architecture is not yet available or developed. However, the modeling approach assumes that the design decisions about the communication between the node and its environment are available. That is, the information about sampled data read from external sensors; control signals sent to external actuators; and network messages sent and received by such a node, is available. It communicates with its environment by means of Sensor Ports (SPs), Actuator Ports (APs), Network Input Ports (NIPs) and Network Output Ports (NOPs). The model of a black-box node is general enough to encapsulate the internal software architecture of a legacy node that is available for partial or complete reuse. In the former case the same node model is called gray-box node model, whereas it is called white-box node model in the later case. In the case of the gray- and white-box node models, the internal software architectures that are reused from legacy nodes communicate with the environment of the nodes via SPs, APs, NIPs and NOPs.

For example, a distributed embedded system modeled with this approach at the vehicle abstraction level is shown in Fig. 6.

The system contains four black-box models of nodes that communicate with each other via the model of a network. It should be noted that if the partial or complete software architecture from the legacy nodes is encapsulated in the nodes as shown in Fig. 6, the node models represent gray- or white-box node models. The network model shown in the figure contains models of all messages that are sent over the network. Moreover, it contains the models of all the signals that are mapped to each message. In addition, it also contains the model of a Signal Database that contains information about the network protocol; signal-to-message mapping and vice versa; encoding and decoding of data in signals and messages.

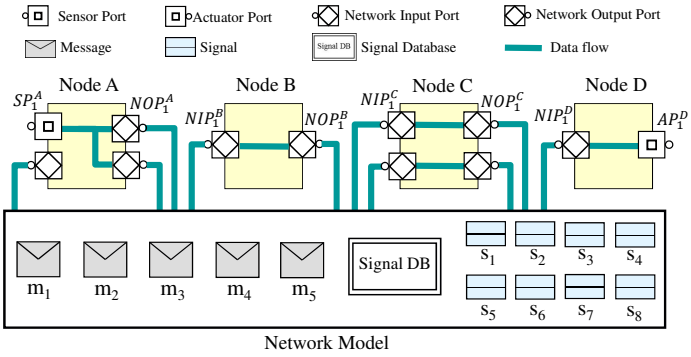


Fig. 6. Model of a network and inter-node communication using the black-box modeling approach.

IV. Extraction of the End-to-end Timing Models

In this section, we discuss the extraction of various models comprising the end-to-end timing model when a distributed embedded system is modeled with the black-box node models.

A. Extraction of the Node Timing Model

The node timing model considered in this work is inspired by the transactional task model [32], [33], [34]. In this model, the tasks are assumed to be scheduled with offsets. The timing model of a node consists of a set of transactions. Each transaction is activated by mutually independent events. This means, the phasing between the events is arbitrary. The activating events can be (1) a periodic sequence of events, or (2) a sporadic sequence of events with a Minimum Inter-arrival Time (MIT) between two consecutive events.

Each transaction contains a number of tasks with each task having an individual priority. A task may not be activated (released for execution) until a certain time, called an *offset*, elapses after the arrival of the event. Associated to each task are the worst-, best- and average-case execution times. A task may also have a jitter that represents variation or difference between the earliest and latest points in time when the task starts to execute. A task may have a blocking time which is the maximum time it has to wait for a resource that is locked by a lower priority task. It can be determined by using a resource sharing protocol, e.g., the Stack Resource Policy [35] or the Priority Ceiling Protocol [36]. Other timing information associated to a task includes precedence relations, trigger dependencies, and real-time requirements such as deadlines.

When the system is developed using the gray- and white-box models of nodes at the vehicle level, the partial or complete internal software architecture of the nodes is available. In such a case the node-level timing information can be extracted by adapting the timing model extraction approaches that are used at the implementation level such as [30]. On the other hand, extraction of timing models from the black-box node models is challenging due to unavailability of the internal software architecture of the node. Intuitively, the existing methods cannot be applied. We extract the node timing model by leveraging on the black-box modeling approach from the previous section. Although response-time analysis cannot be performed based on the extracted timing model, the extracted timing information is sufficient enough to perform end-to-end timing analysis with relatively low precision. It should be noted that the precision of the analysis is higher in the case of gray- and white-box models of nodes.

1) Extraction of Control Flows: Since the end-to-end timing analysis is performed on distributed chains consisting of tasks and messages, the control flows within these chains should be extracted. By control flow we mean how each task or message within the chain is activated. When a distributed embedded system is modeled with the black-box node models, the control flow is extracted from the node ports. There are three pieces of information associated to control flow that must be extracted from each node port. The first information concerns the source of triggering, i.e., whether the port is triggered by another port or by an independent trigger source (e.g., clock, event or interrupt). The second information specifies if the trigger source is periodic or sporadic. Finally, the third information concerns the period or MIT of the trigger source.

In order to extract such triggering information, we associate a *Trigger* parameter with each node port. We attach three attributes with this parameter namely *Trigger.source*, *Trigger.type* and *Trigger.value*. If a port is triggered by an independent trigger source then the corresponding *Trigger.source* attribute is assigned "independent", e.g., all node ports except AP_2 and NOP_3 in Fig. 7. Whereas, if a port is triggered by another port then the *Trigger.source* attribute is assigned "dependent", e.g., AP_2 and NOP_3 in Fig. 7. If a node port is triggered by a clock, the corresponding *Trigger.type* attribute is assigned "periodic" while the *Trigger.value* attribute gets the clock period. For example SP_1, NIP_1, NIP_3, AP_1, NOP_1 and AP_2 in Fig. 7. However, if a node port is triggered by a sporadic event or interrupt, the corresponding *Trigger.type* attribute is assigned "sporadic" while the *Trigger.value* attribute gets the corresponding MIT. For example NIP_2, SP_2, NOP_2, and NOP_3 in Fig. 7. It should be noted that a periodic or sporadic source for sensor and network input ports corresponds to polling- or interrupt-based sampling or message receiving routines respectively.

2) Extraction of Data Flows: In a black-box node model, the data flow refers to the way that the data connections are established within the node between its input ports (SPs and NIPs) and output ports (APs and NOPs). There can only be three sources of data that are produced at each output port.

(i) The data is initiated and produced at the node output port, i.e., the data is not associated to any NIP or SP. In this case, the data producer would correspond to the SWC directly connected to the port as if the internal software architecture were available. However, this SWC is not available in the black-box node model. This scenario is depicted in Fig. 8 where AP_1 and NOP_1 do not possess a data connection with any other port within the node.

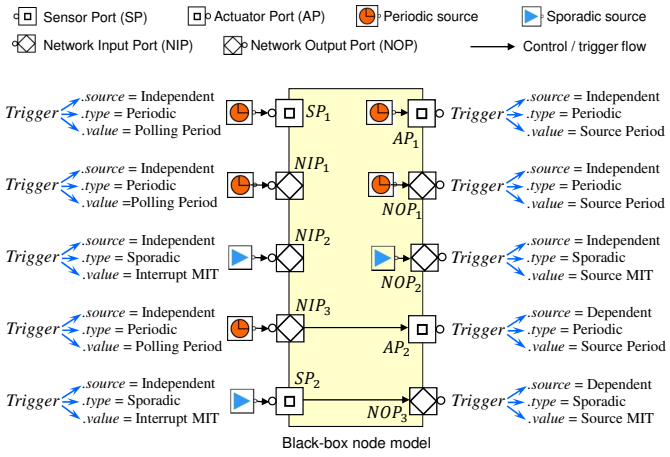


Fig. 7. Extraction of control flows from the black-box node model.

(ii) The data arrives from one of the SPs belonging to the same node. In this case, the node output port is connected to the corresponding SP. This situation is shown in Fig. 8 where AP_2 and NOP_2 receive data directly from SP_1.

(iii) The data arrives from the network through one of the NIPs belonging to the same node. In this case, the node output port is connected to the NIP within the node. This scenario is shown in Fig. 8 where AP_3 and NOP_3 receive data directly from NIP_1.

In order to extract the data flow, each AP and NOP is associated with the *Data_Path(Port)* attribute. This attribute can get one of the three values as shown below.

$$Data\_Path(Port) = \{INIT|SP|NIP\}$$

INIT means that the data is initiated at the corresponding AP or NOP, e.g, AP_1 and NOP_1. On the other hand, *SP* and *NIP* means that the data at the output port arrives from one of the SPs or NIPs as shown in Fig. 8.

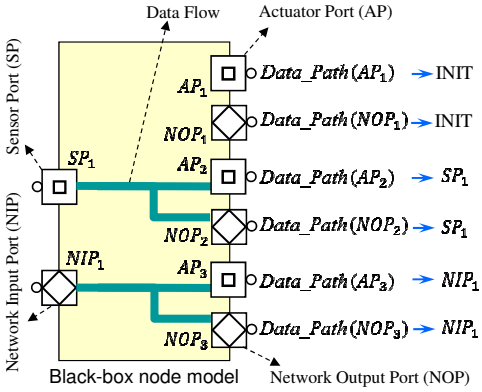


Fig. 8. Extraction of data flows within the black-box node model.

3) Extraction of the Internal Delays in a Node: In order to improve the precision of end-to-end timing analysis of the black-box node models, we introduce a delay object to capture the jitter received at each node port. If there is a possibility to predict or estimate the internal delays in the black-box models, they can be specified on the delay objects which can be connected to the node ports. For example, such a delay object connected to a NOP represents the difference between the worst- and best-case response times of the task corresponding to the software component that would be directly connected to the NOP once the internal software architecture of the node is available. Similarly, such delays can be captured from the NIPs, sensor and actuator ports. If these delays are specified and subsequently extracted in the timing model, the analysis engines take them into account and provide the timing analysis results with a better precision.

B. Extraction of the Network Timing Model

Most of the timing information about network messages can be extracted from the attributes that are explicitly specified on the model of a message that is shown in Fig. 6. The message model is general as it allows the specification of both priority and ID. However in order to be unambiguous in some protocols such as CAN, where message priorities are unique and equal to corresponding IDs, the analysis engines extract the message priority from its ID. The size of a message can be equal to or higher than the size of the frame in the message model. However, a one-to-one mapping is considered between a message and a frame in the case of CAN and its higher-level protocols. This means, we consider the message size such that it is able to fit one CAN frame. Some of the timing information about the messages should be either extracted from the modeled application or calculated/estimated based on other attributes. For instance, the worst-case transmission time is calculated based on the Data Length Code (DLC) parameter and the network speed specified on the model of the network as shown in Fig. 6.

The transmission type of each message is extracted from the *Trigger.type* attribute associated to the NOP that queues the message for transmission. Similarly, depending upon the transmission type, the period or MIT of a message is extracted from the *Trigger.value* attribute associated to the NOP that queues the message for transmission. Another timing parameter is message jitter that is generally inherited from the task corresponding to the sender software component. Since this information is not available in the black-box model, this parameter is inherited from the delay specified on the delay object that is connected to the corresponding NOP. In the case of a data connection between the NOP and one of the NIPs or SPs within the same node, this delay corresponds to the time elapsed from the triggering of the NIP or SP to the triggering of the NOP. If this delay is not specified, its default minimum value can be considered as 1 μs that corresponds to the transmission time of one bit of data on the CAN bus at its maximum operating speed of 1 Mbps (currently, we don't consider CAN-FD [37]). It should be noted that the precision of the analysis results partly depend upon the accuracy of the specified delay.

C. Extraction of the Linking Model

The linking model captures the end-to-end real-time requirements and linking information within each distributed chain that is composed of components, messages, data flow and control flow. The real-time requirements such as end-to-end deadlines, age and reaction constraints are specified at the system level, i.e., outside of the node models. If a distributed embedded system is modeled using the gray- and white-box node models, such constraints specified on distributed chains

may reside either inside or outside of the nodes. However, these constraints must be specified outside of the black-box node if it is located along the path of the distributed chain under analysis. Each such constraint has a start object and an end object. The real-time requirements are extracted from the end object of each constraint. The placement of these objects determine the length of the distributed chain to analyze.

1) Extraction of Distributed Chains: The references to all the NIPs, NOPs, SPs, APs and messages along each distributed chain are collected in a data structure denoted by *Constrained_Chain(ID, type)*. Where, ID denotes a unique identification of the chain on which a timing constraint is specified. Whereas, type denotes the type of timing constraint that is specified on the chain. For example, assume that the start and end objects for the age constraint are specified before the sensor port belonging to Node A (denoted by SP_1^A) and after the actuator port belonging to Node D (denoted by AP_1^D) respectively in Fig. 6. This represents a distributed chain whose input data is sensed from the sensor port in Node A, while the corresponding output appears at the actuator port of Node D after traversing through Node B, Node C and the network. The sequence of ports and messages captured in the linking model from the corresponding *Constrained_Chain(1, Age)* in Fig. 6 is given below.

$$\begin{aligned} Constrained\_Chain(1, Age) := & \; SP_1^A \rightarrow NOP_1^A \rightarrow \mathrm{m}_1 \rightarrow \\ & NIP_1^B \rightarrow NOP_1^B \rightarrow \mathrm{m}_2 \rightarrow \\ & NIP_1^C \rightarrow NOP_1^C \rightarrow \mathrm{m}_5 \rightarrow \\ & NIP_1^D \rightarrow AP_1^D \end{aligned}$$

In order to extract the data flows in each *Constrained_Chain(ID, type)*, one of the three possible sources of data produced at the NOP or AP within a node along the chain is captured from the corresponding *Data_Path(Port)* attribute. The data connections of each message from the sender NOP and receiver NIPs are extracted from the signal database object as shown in Fig. 6. Moreover, the mapping, encoding, packing and decoding information of signals from/to messages along a *Constrained_Chain(ID, type)* is also extracted from the signal database. The control flow along each distributed chain is captured by extracting the *Trigger.dependency*, *Trigger.type* and *Trigger.value* attributes associated to each port along *Constrained_Chain(ID, type)*. These attributes for each NOP capture the trigger flows related to network messages. The size of data arriving from a sensor or delivered to an actuator is assumed to be equal to 1 byte if it is not specified on the optional parameter associated to each sensor or actuator port respectively.

2) Extraction of Mode-related Information: Based on the requirements specified on the system, each node may have more than one mode. In essence, a mode is an application of its own. The system may switch from one mode to another during its execution. In the case of the black-box node model, we assume that there is only one mode because the internal software architecture is not available. However, the gray- and white-box node models may have more than one mode. This complicates the extraction of distributed chains that span over more than one node if the nodes along the chain have more than one mode. For example, assume that the distributed embedded system as shown in Fig. 6 is modeled with the white-box node models and that there are 2, 2, 3, and 2 modes in Node A, Node B, Node C and Node D respectively. Also assume that the age constraint is specified in a similar fashion as in Section IV-C1. There are 24 ($2 * 2 * 3 * 2 = 24$) different distributed chains that are extracted in this case. All these extracted chains correspond to the single *Constrained_Chain(1,Age)* in Section IV-C1 if only the black-box node models are used in Fig. 6.

V. Support for End-to-end Timing Analysis

When a distributed embedded system is developed using the black-box models of nodes, the internal software architecture of nodes is not available. Intuitively, timing properties of the tasks are also not available. Hence, the response times of tasks cannot be calculated. However, most of the timing information about the network communication is assumed to be available, therefore, response times of messages can be calculated. The end-to-end response times of distributed chains depend upon the timing properties of all tasks and messages. Due to lack of complete timing information about distributed chains, the calculated end-to-end response times may not be of significant values, especially in the case where internal delays connected to the node ports (see Section IV-A3) are not specified.

On the other hand, the age and reaction delays can be calculated with relatively higher precision. This is because the contribution of clock periods along the chain have the lion's share in the delays compared to the contribution of the response times of tasks along the distributed chain [9], [26]. For example, the end-to-end response time of the chain in Fig. 5 is 3 time units (for simplicity, we do not consider any interference other than the one imposed by the two tasks on each other). Whereas, the age delay is 24 time units. Although the reaction delay is equal to the end-to-end response time in this case, the delay can be significantly higher if the number of tasks along the chain increases. In conclusion, the calculations for the age and reaction delays in the case of black-box models is valuable because the calculated delays can be used to provide initial validation or refinement of the system model with respect to timing requirements.

The end-to-end path-delay analysis of the system modeled with the black-box node models can be performed using Algorithm 1. The algorithm is adapted from the existing analysis [26], [9]. The purpose of this algorithm is not to revisit the existing analysis [26] but to show how it can be adapted to calculate the age and reaction delays when black-box models of nodes are used. The number of timed paths can be determined in relation to the times when each port along the chain is triggered; instead of determining it from the activation and execution times of the tasks along the chain. For example, the first triggering of a port is assumed to be equivalent to the activation of the first instance of the task corresponding to the software component that would be connected to the port if the internal software architecture of the node is available.

It can be seen on line 4 in Algorithm 1 that the age delay in a timed path not only depends upon the response time of the current instance of the last task but also on the activation times of the current instances of the first and last tasks in the chain. The calculations for the reaction delay depend upon the last in first out delay; the activation time of the current instance of the first task; and the activation time of the previous non-written instance of the first task belonging to the previous valid

Algorithm 1 Algorithm for end-to-end path-delay analysis of the systems developed using the black-box node models.

1: **begin**
2: FIND_ALL_VALID_TPS() ▷ TP: Timed Path
3: **procedure** COMPUTE_AGE$_{delay}$(LL_TP) ▷ LL: Last in Last out
4: Age$_{delay}$ = α_n(inst) + δ_n(inst) - α_1(inst) ▷ α_n(inst) and δ_n(inst): Activation and response times of the corresponding instance "inst" of nth task in TP respectively.
5: **return** Age$_{delay}$
6: **end procedure**
7: **procedure** COMPUTE_LF$_{delay}$(LF_TP) ▷ LF: Last in First out
8: LF$_{delay}$ = α_n(inst) + δ_n(inst) - α_1(inst)
9: **return** LF$_{delay}$
10: **end procedure**
11: **procedure** COMPUTE_REACTION$_{delay}$(FF_TP) ▷ FF: First in First out
12: Reaction$_{delay}$ = LF$_{delay}$ + α_1(inst) - $\alpha_{1,Pred}$(inst) ▷ $\alpha_{1,Pred}$(inst): Activation time of the previous non-overwritten instance of nth task in TP.
13: **return** Reaction$_{delay}$
14: **end procedure**
15: **for all** *Constrained_Chain(ID, type)* **do**
16: $Age \leftarrow 0$, $Reaction \leftarrow 0$ ▷ Initialization
17: $Age\_TP_{count} \leftarrow$ GET_ALL_LL_TPS()
18: $Reaction\_TP_{count} \leftarrow$ GET_ALL_FF_TPS()
19: **for** i:=1 to LL_TP_{count} **do**
20: **if** COMPUTE_AGE$_{delay}$(i) $> Age$ **then**
21: $Age \leftarrow$ COMPUTE_AGE$_{delay}$(i) ▷ Age delay is the maximum delays among all LL TPs.
22: **end if**
23: **end for**
24: **for** i:=1 to FF_TP_{count} **do**
25: **if** COMPUTE_REACTION$_{delay}$(i) $> Reaction$ **then**
26: $Reaction \leftarrow$ COMPUTE_REACTION$_{delay}$(i) ▷ Reaction delay is the maximum delay among all FF TPs.
27: **end if**
28: **end for**
29: **end for**
30: **end**

timed path. Since some of these parameters are not available per se in the black-box models, the response time of a task is assumed to be equal to the delay specified on the delay object connected to the corresponding NIP, NOP, sensor or actuator port (discussed in Section IV-A3). Similarly, the activation time of the task can be extracted from the activation time of the clock which is directly connected to the corresponding port. Using these assumptions and Algorithm 1, the existing analysis is applicable to the system that is developed with the black-box models of nodes. It should be noted that the algorithm is also applicable to the system that is modeled with the gray- and white-box node models. However in such a case, depending upon the number of modes in each node, several *Constrained_Chains(ID,type)* corresponding to one end-to-end timing constraint are analyzed. In this case, the analysis results not only include the end-to-end delays for each combination of distributed chains due to multiple modes, but also the worst-case end-to-end delay among all combinations.

VI. Proof of Concept and a Case Study

In order to show the proof of concept, we implement the proposed timing model extraction approach in the existing industrial tool suite Rubus-ICE. Moreover, the adapted end-to-end timing analysis algorithm is implemented on top of the existing end-to-end timing analysis plug-in in Rubus-ICE [9]. The existing analysis supports the analysis at the implementation level. With the extensions, the plug-in now supports the analysis at the vehicle level. In order to show the applicability of our approach, we conduct a vehicular case study by using the extended implementation in Rubus-ICE.

A. Modeling of an Adaptive Cruise Control System using Black-box and Legacy Models

We mimic a typical industrial scenario by reusing most of the artifacts from earlier releases or previous iterations during the development of a vehicular distributed embedded system. Intuitively, most of the design decisions about network communication can be made early during the development. The objective is to model and analyze an Adaptive Cruise Control (ACC) system using previously developed models of the Cruise Control (CC) system at the vehicle level. The CC system allows a vehicle to automatically maintain a steady speed to the preset speed. It controls the engine throttle depending upon the velocity feedback from the speed sensor. The ACC system is an extension of the CC system such that it provides the cruise control functionality by adapting itself to the traffic. It uses proximity sensors such as 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 it accelerates the vehicle to match the preset speed.

The legacy CC system that is available for reuse has been modeled with four nodes namely Cruise Control ($Node_{CC}$), Engine Control ($Node_{EC}$), Brake Control ($Node_{BC}$) and User Interface ($Node_{UI}$). The nodes communicate among each other via one CAN bus. The ACC system can be modeled by reusing models of four nodes from the CC system and introducing a fifth node denoted by Adaptive Cruise Control ($Node_{ACC}$) that provides the adaptive functionality. We assume that $Node_{ACC}$ communicates only with $Node_{CC}$ via the CAN bus. Hence, we consider only $Node_{ACC}$ and $Node_{CC}$ for simplicity. We also assume that the internal software architecture of $Node_{CC}$ can be partially reused. Hence, it is modeled with a gray-box node model. On the other hand, $Node_{ACC}$ is modeled as a black-box node model.

The model of the ACC system in Rubus-ICE is shown in Fig. 9. The system consists of a black-box node model $Node_{ACC}$, a gray-box node model $Node_{CC}$ and a CAN bus. The speed of the CAN bus is 250 Kbit/s. The model of CAN contains eight messages whose models and attributes are shown in Fig. 10. The model of signal database that encapsulates the signal-to-message mapping and vice versa is shown in the figure. There are three NIPs in $Node_{ACC}$ for receiving three CAN messages containing vehicle speed, RPM value and status of manual brake respectively. We assume that

the delays associated to these NIPs cannot be estimated. It has one sensor port that receives sampled radar signals. We assume that a delay object is connected to the sensor port with an estimated delay of 500 μs. The main purpose of this node is to calculate the control information regarding presence of other vehicles in its proximity. It has only one NOP that sends a CAN message, carrying proximity control information, to $Node_{CC}$. We assume that a delay object is connected to the NOP with an estimated delay of 700 μs. All the node ports are triggered independently by periodic clocks, each having a period of 10 ms.

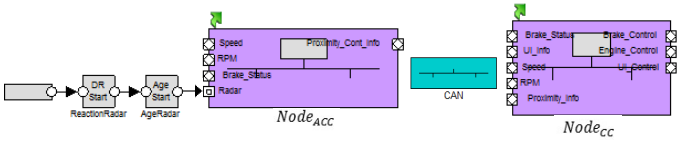
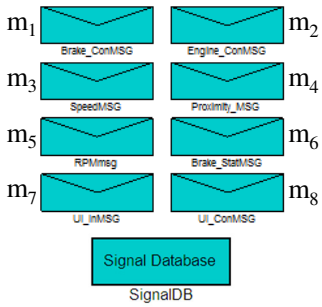


Fig. 9. Model of Adaptive Cruise Control at vehicle level in Rubus-ICE.



Msg	DLC	Prio	Period
m_1	6	0	10 ms
m_2	6	1	10 ms
m_3	4	2	10 ms
m_4	8	3	10 ms
m_5	4	4	10 ms
m_6	2	5	10 ms
m_7	8	6	10 ms
m_8	8	7	10 ms

Fig. 10. Models of CAN messages in Rubus-ICE.

The gray-box model of $Node_{CC}$ as shown in Fig. 9 has five NIPs for receiving five CAN messages containing vehicle speed, RPM value, status of manual brake, user interface and proximity information respectively. The main purpose of this node is to compute the control information that is used to adjust the speed of the vehicle with respect to the cruising speed or clearing distance from the preceding vehicle. There are three NOPs that send CAN messages carrying control information about engine actuation, brake actuation and user interface control. The internal software architecture of $Node_{CC}$ that is partially reused is shown in Fig. 11. It contains only one mode. There are five SWCs: one performs input processing; one carries out the computation of control information; while the remaining SWCs send CAN messages carrying engine, brake and user interface control information. The WCETs of these SWCs are 200 μs, 180 μs, 80 μs, 80 μs and 80 μs respectively.

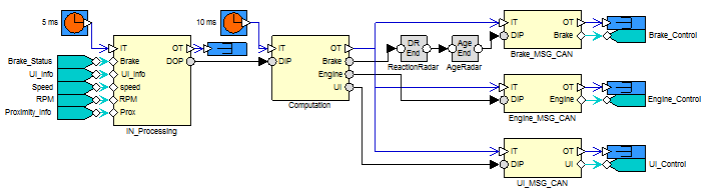


Fig. 11. Software architecture of gray-box model of $Node_{CC}$ in Rubus-ICE.

B. End-to-end Timing Analysis

We are interested in the calculations of end-to-end path delays from the arrival of radar sensor values at $Node_{ACC}$ until production of brake control signals in $Node_{CC}$. We specify the start objects for the age and reaction constraints namely AgeRadar and ReactionRadar at the sensor port of $Node_{ACC}$ in Fig. 9. Whereas the end objects for the age and reaction are specified in $Node_{CC}$ as shown in Fig. 11. The values of the specified age and reaction constraints are 20 ms and 30 ms respectively. Accordingly, the extracted distributed chain to be analyzed under the specified timing constraints consists of node ports, messages and SWCs (only in $Node_{CC}$) is as follows.

$$Constrained\_Chain(1, Age\&Reac) := Radar^{Node_{ACC}} \rightarrow Proximity\_Cont\_Info^{Node_{ACC}} \rightarrow \mathrm{m}_4 \rightarrow Proximity\_Info^{Node_{CC}} \rightarrow IN\_Processing\_SWC^{Node_{CC}} \rightarrow Computation\_SWC^{Node_{CC}}$$

The response times of messages m_1, m_2, m_3, m_4, m_5, m_6, m_7 and m_8 calculated by the analysis engines are 1620 μs, 2000 μs, 1840 μs, 3880 μs, 2760 μs, 3060 μs, 3600 μs and 4600 μs respectively. Whereas, the calculated network utilization is 36%. The age and reaction delays calculated by the analysis engines are equal to 13960 μs and 23960 μs.

C. Discussion

It can be seen that the age and reaction delays meet the specified end-to-end deadlines of 20 ms and 30 ms respectively. This means that the specified constraints are good enough to be used at the lower abstraction levels. It should be noted that the internal delays associated with the node ports in the black-box node models are assumed to be based on judgements by the industrial experts. If it can be estimated that the specified delays are sufficient enough to correspond to WCETs of the SWCs that are connected to the corresponding node ports, the end-to-end timing requirements can be refined based on the calculated delays.

On the contrary, if the age and reaction constraints are not met then some useful information can be provided which can help the user to perform a refinement of models and timing requirements as follows.

(i) The periods of independent clocks along a distributed chain have a significant impact on the path delay. It may or may not be possible to change the clock periods in legacy nodes. However, reducing the periods associated to the node ports in black-box nodes could reduce the age and reaction delays. Such modifications could be more helpful in the case of under-utilized nodes.

(ii) Another way to lower the delays is by decreasing the number of independent clocks along the chain. For example, the age and reaction delays may be lowered if $Proximity\_Cont\_Info^{Node_{ACC}}$ NOP is triggered by $Radar^{Node_{ACC}}$ SP instead of an independent clock. This information may be useful at a lower abstraction level or at a later stage during the development when the internal software architecture of the black-box node is modeled.

(iii) It may be possible that the internal delays associated with the node ports in the black-box node models are over-estimated. Reducing these estimations can lower the age and reaction delays.

(iv) It may also be the case that end-to-end timing requirements specified by the user are very tight. By making a comparison with the calculated age and reaction delays, the user may be able to refine the timing requirements.

VII. Conclusion

In this paper we have introduced a new approach for the extraction of end-to-end timing models from vehicular distributed embedded systems. The proposed approach is applicable at a high level of abstraction where the system contains crude models of nodes that lack their internal software architectures. Moreover, the system may also contain legacy nodes whose internal software architectures can be partly or fully reused. We have also provided an adaptation to the existing analysis engines to support the end-to-end timing analysis of these systems at a high abstraction level and at an earlier phase during their development. The analysis results can provide guidelines to perform optimizations; improve timing requirements; and do model refinements at such a high level of abstraction. In order to provide a proof of concept, we have implemented the timing model extraction approach and have adapted analysis in the existing industrial tool suite Rubus-ICE. Using the extended tool we have conducted a vehicular-application case study to show the applicability of the proposed approach.

The timing model extraction approach is applicable to any component technology that uses a pipe-and-filter style for components interconnection and supports separation between control and data flows. We believe, the tools implementing the timing model extraction approach and adapted analysis may prove helpful for the software development organizations in the vehicular domain to decrease the costs for development and testing.

In the future, we plan to conduct a comprehensive case study on a use case from our industrial partners. The aim is to model and timing analyze a vehicular application consisting of multiple networks (e.g., CAN and Switched Ethernet) or network segments (e.g., CAN-CAN) at the vehicle level. Another interesting future work is to perform sensitivity analysis on end-to-end timing of a vehicular distributed embedded system that is modeled with only (a) the white-box node models, (b) the gray-box node models, or (c) the black-box node models.

Acknowledgement

The work in this paper is supported by the Swedish Foundation for Strategic Research (SSF) within the project PRESS; the Swedish Knowledge Foundation (KKS) within the project FEMMVA; and ARTEMIS within the project CRYSTAL. We thank our industrial partners Arcticus Systems AB and Volvo AB Sweden.

References

[1] Peter Thorngren, keynote Talk: Experiences from EAST-ADL Use, EAST-ADL Open Workshop, Gothenberg, Oct., 2013.
[2] T. A. Henzinger and J. Sifakis, "The Embedded Systems Design Challenge," in *14th International Symposium on Formal Methods (FM), Lecture Notes in Computer Science*. Springer, 2006, pp. 1–15.
[3] I. Crnkovic and M. Larsson, *Building Reliable Component-Based Software Systems*. Norwood, MA, USA: Artech House, Inc., 2002.
[4] "AUTOSAR Techincal Overview, Release 4.1, Rev. 2, Ver. 1.1.0., The AUTOSAR Consortium, Oct., 2013," http://autosar.org.
[5] "TIMMO Methodology, Version 2," *TIMMO (TIMing MOdel), Deliverable 7*, October 2009, The TIMMO Consortium.
[6] "TIMMO-2-USE," https://itea3.org/project/timmo-2-use.html.
[7] CRYSTAL - CRitical sYSTem engineering AcceLeration, http://www.-crystal-artemis.eu, accessed May, 2014.
[8] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocess. Microprogram.*, vol. 40, pp. 117–134, April 1994.
[9] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "Support for end-to-end response-time and delay analysis in the industrial tool suite: Issues, experiences and a case study," *Computer Science and Information Systems*, vol. 10, no. 1, 2013.
[10] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "Translating timing constraints during vehicular distributed embedded systems development," in *1st International Workshop on Model-Driven Engineering for Component-Based Software Systems*, September 2014.
[11] A. Bucaioni, S. Mubeen, A. Cicchetti, and M. Sjödin, "Exploring Timing Model Extractions at EAST-ADL Design-level Using Model Transformations," in *International Conference on Information Technology: New Generations (ITNG)*, April 2015.
[12] "EAST-ADL Domain Model Specification, V2.1.12.," http://www.east-adl.info/Specification/V2.1.12/EAST-ADL-Specification_V2.1.12.pdf.
[13] A. Hamann, R. Henia, R. Racu, M. Jersak, K. Richter, and R. Ernst, "Symta/s - symbolic timing analysis for systems," 2004.
[14] "Arcticus Systems AB," http://www.arcticus-systems.com.
[15] K. Hänninen et.al., "The Rubus Component Model for Resource Constrained Real-Time Systems," in *3rd IEEE International Symposium on Industrial Embedded Systems*, June 2008.
[16] Timing Augmented Description Language (TADL2) syntax, semantics, metamodel Ver. 2, Deliverable 11, Aug. 2012.
[17] TADL: Timing Augmented Description Language, Del. 6, Oct. 2009.
[18] "Rubus models, methods and tools," http://www.arcticus-systems.com.
[19] "Rubus ICE-Integrated Development Environment," http://www.-arcticus-systems.com.
[20] "The UML Profile for MARTE: Modeling and Analysis of Real-Time and Embedded Systems," January 2010. [Online]. Available: http://www.omgmarte.org/
[21] TIMMO-2-USE Methodology Description, Ver. 2, Del. 13, July. 2012.
[22] X. Ke, K. Sierszecki, and C. Angelov, "COMDES-II: A Component-Based Framework for Generative Development of Distributed Real-Time Control Systems," in *13th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, Aug. 2007.
[23] S. Sentilles, A. Vulgarakis, T. Bures, J. Carlson, and I. Crnkovic, "A Component Model for Control-Intensive Distributed Embedded Systems," in *11th International Symposium on Component Based Software Engineering (CBSE2008)*. Springer, October 2008, pp. 310–317.
[24] A. Ohno, T. Azumi, and N. Nishio, "TECS components providing functionalities of OSEK specification for ITRON OS," *Journal of Information Processing*, vol. 22, no. 4, pp. 584–594, 2014.
[25] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "Towards Extraction of Interoperable Timing Models from Component-Based Vehicular Distributed Embedded Systems," in *International Conference on Information Technology: New Generations (ITNG)*, April 2014.
[26] N. Feiertag, K. Richter, J. Nordlander, and J. Jonsson, "A Compositional Framework for End-to-End Path Delay Calculation of Automotive Systems under Different Path Semantics," in *Workshop on Compositional Theory and Technology for Real-Time Embedded Systems*, dec. 2008.
[27] pyCPA Tool, http://pycpa.readthedocs.org/, accessed Mar. 2015.
[28] CANalyzer, Ver. 8.5, http://www.vector.com/vi_canalyzer_en.html, accessed Mar. 2015.
[29] MAST–Modeling and Analysis Suite for Real-Time Applications, http://mast.unican.es/, accessed Mar. 2015.
[30] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "Communications-Oriented Development of Component- Based Vehicular Distributed Real-Time Embedded Systems," *Journal of Systems Architecture*, vol. 60, no. 2, pp. 207–220, 2014.
[31] S. Mubeen, M. Sjödin, T. Nolte, J. Lundbäck, M. Gålnander, and K.-L. Lundbäck, "Modeling of Legacy Distributed Embedded Systems at Vehicle Abstraction Level," Mälardalen University, Technical Report, April 2015, http://www.es.mdh.se/publications/3892-.
[32] K. Tindell, "Adding Time-Offsets to Schedulability Analysis," Department of Computer Science, University of York, Tech. Rep., Jan. 1994.
[33] J. Palencia and M. G. Harbour, "Schedulability Analysis for Tasks with Static and Dynamic Offsets," *Real-Time Systems Symposium*, 1998.
[34] J. Mäki-Turja and M. Nolin, "Efficient implementation of tight response-times for tasks with offsets," *Real-Time Syst.*, vol. 40, no. 1, pp. 77–116, 2008.
[35] T. P. Baker, "Stack-based scheduling for realtime processes," *Real-Time Systems*, vol. 3, no. 1, pp. 67–99, Apr. 1991.
[36] L. Sha, R. Rajkumar, and J. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization," *IEEE Transactions on Computers*, vol. 39, no. 9, pp. 1175–1185, 1990.
[37] Robert Bosch GmbH, "CAN with Flexible Data-Rate (CAN FD), White Paper, Ver. 1.1." 2011.