Supporting Early Modeling and End-to-end Timing Analysis of Vehicular Distributed Real-Time Applications

Saad Mubeen*, Mikael Sjödin* and Jukka Mäki-Turja*†
* Mälardalen University, Sweden. † Arcticus Systems, Järfälla, Sweden
{saad.mubeen, mikael.sjodin, jukka.maki-turja}@mdh.se

Abstract—**The current model- and component-based development approaches for automotive distributed real-time systems have non-existing, or limited, support for modeling network traffic originating from outside the vehicle, i.e., vehicle-to-vehicle, vehicle-to-infrastructure, and cloud-based applications. We present novel modeling and analysis techniques to allow early end-to-end timing analysis of distributed applications based on their models and simple models of network traffic that originates from outside of the model. As a proof of concept, we implement these techniques in the existing industrial tool suite Rubus- ICE which is used for the development of software for vehicular embedded systems by several international companies. We also conduct an application-case study to validate our techniques.**

I. Introduction

An important class of emerging distributed applications is novel functionality in road vehicles. These applications realize novel services based on vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communications. Both V2V and V2I are expected to support novel applications for road-safety, traffic efficiency, and driver/passenger comfort and entertainment. Already today there are examples of traffic related cloud-services, e.g., community map and turn-by-turn navigation such as Waze [1] and traffic-congestion information by Google. However, to be successfully adopted, the development of these new applications needs to be integrated in contemporary workflow for development of vehicular functions.

Development strategies for automotive, and other vehicular applications, are to a great extent based on model-based development. In these approaches detailed models of the behavior of each on-board function is developed and successively refined to reach the implementation of each node. In this refinement process, the communications needed for each node are derived and a message set for each on-board network is defined. Moreover, timing parameters and requirements for each message are established. In its current form there is often non-existing, or limited, support to model network traffic originating from outside the vehicle. That is traffic from V2V, V2I, and other, e.g., cloud-based applications are not naturally modeled and analyzed in existing approaches.

A. Goals and Paper Contributions

We present novel modeling and analysis techniques to allow early end-to-end timing analysis of distributed applications based on their models and simple models of bus traffic that originates from outside of the model. This "outside traffic" could come from the mentioned external services, from legacy nodes that lack proper behavior models, or from crude preliminary models of nodes that have not been completely modeled yet. Regardless of source, it is important to, at an early stage be able to analyze end-to-end timing behavior of vehicle internal functions, while taking into account the outside generated traffic. In this context, the term early refers to the stage before system development where the traffic model can be derived from the functional models and their planned allocations.

We extend our previous modeling and analysis methods [2], [3] to support modeling of outside messages and end-to-end timing analysis of the system containing these messages at an early stage during the development. As a proof of concept, we implement the extended method in the existing industrial tool suite Rubus- ICE [4], [5], [3] which is used for the development of software for vehicular embedded systems by several international companies [6], [7], [8], [9]. We also conduct an application-case study to validate our methods.

B. Paper Layout

In Section II, we discusses the research problem. Section III presents the background and related work. Section IV discusses the proposed solution. Section V presents the case study. Section VI concludes the paper.

II. Problem Statement

Often, the vehicular distributed real-time systems consist of ECUs which are provided by various suppliers. For each ECU, the software may be developed either in-house, reused from the product family or bought as COTS (Commercial Off-The-Shelf). The problem arises when the requirements dictate the modeling and end-to-end timing analysis of a component-based distributed real-time system at an early stage during the development. At this stage, the models of some ECUs may not be available. However, the signals and messages which these missing ECUs are supposed to send and receive have been decided. In such a system, the network is assumed to contain messages whose sender nodes are not developed yet. Similarly, the available ECUs may send messages via network to the nodes that will be available at a later stage.

Some reasons behind the requirement for early timing analysis are to support design space exploration, allow fine tuning of the system with respect to real-time requirements and detect timing errors as early as possible. Late detection of timing errors can be very expansive in terms of time and cost. The detection of timing errors in a real-time system after its release to the market is a financial and production disaster.

In order to ensure that a distributed real-time system will behave in a timely manner during its execution, we need to analyze tasks, messages and event chains in distributed transactions and predict the end-to-end delays. To perform

the end-to-end timing analysis of such a system, there are several problems that need to be solved. One such problem is to provide a support in the component technology to model "outside traffic" by means of stand-alone messages. Another problem is the extraction of timing information from these systems. There exist timing dependencies among messages and their sender and receiver tasks (that may not exist at this stage).

A message inherits some timing properties from its sender task, e.g., transmission type and period. If the sender task is triggered periodically then the message it sends is also periodic. Further, the message inherits period from its sender task. Similarly, if the sender task is activated sporadically then the corresponding message is sporadic and the message inherits inhibit time from the sender task. In the case of mixed transmission mode, the message is mixed [10] and inherits both period and inhibit time from its sender. When distributed real-time systems are analyzed, each message is assumed to inherit the response time of the sender task as its release jitter (attribute inheritance [11]). For the messages whose sender tasks are unknown (because the sender ECUs are not available yet or the network traffic is generated from outside of the model), these properties must be extracted in the timing model. Otherwise, the end-to-end timing analysis cannot be performed.

III. Background and Related Work

A. 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 [4] 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 [6], [7], [8], [9]. The Rubus concept is based around the Rubus Component Model (RCM) and its development environment Rubus-ICE (Integrated Component development Environment) [4], 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.

1) The Rubus Component Model: RCM expresses the infrastructure for software functions, i.e., the interaction between the software functions in terms of data and control flow separately. The control flow is expressed by triggering objects such as clocks and events as well as other components. In RCM, the basic component is called Software Circuit (SWC). The execution semantics of an SWC 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.

2) The Rubus Code Generator and Run-Time System: From the resulting architecture of connected SWCs, functions are mapped to run-time entities; tasks. Each external event trigger defines a task and SWCs connected through the chain of triggered SWCs (trigger chain) are allocated to the corresponding task. All clock triggered "chains" are allocated to an automatically generated static schedule that fulfills the precedence order and temporal requirements. Within trigger chains, inter-SWC communication is aggressively optimized to use the most efficient means of communication possible for each communication link. Allocation of SWCs to tasks and construction of schedule can be submitted to different optimization criterion to minimize, e.g., response times for different types of tasks, or memory usage. The run-time system executes all tasks on a shared stack, thus eliminating the need for static allocation of stack memory to each individual task.

3) The Rubus Analysis Framework: The Rubus model allows expressing real-time requirements and properties at the architectural level. For example, it is possible to declare real-time requirements from a generated event and an arbitrary output trigger along the trigger chain. For this purpose, the designer has to express real-time properties of SWCs, such as Worst Case Execution Times (WCETs) 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 model supports distributed end-to-end response time and delay analysis and shared stack analysis.

B. Related Work

There are very few commercial component models for distributed real-time systems especially in automotive domain. In our previous work, we carried out a detailed comparison of RCM with various models for distributed real-time systems [2]. We briefly highlight a few of them.

AUTOSAR (AUTomotive Open System ARchitecture) [12] is a standardized software architecture for the development of software in automotive domain. It can be viewed as a standardized distributed component model. When AUTOSAR was being developed, there was no focus placed on its ability to specify and handle real-time requirements and properties. On the other hand, such requirements and capabilities were strictly taken into account right from the beginning during the development of RCM. AUTOSAR describes embedded software development at a relatively higher level of abstraction compared to RCM. A Software Circuit in RCM more resembles to a runnable entity (a schedulable part of AUTOSAR software component) instead of AUTOSAR software component. As compared to AUTOSAR, RCM clearly distinguishes between control flow and data flow among software components in a node. AUTOSAR hides the modeling of execution environment. On the other hand, RCM explicitly allows the modeling of execution requirements, e.g., jitter and deadlines, at an abstraction level close to the functional modeling while abstracting the implementation details.

TIMMO (TIMing MOdel) [13] is an initiative to provide AUTOSAR with a timing model. It describes a predictable methodology and a language, called TADL [14], to express timing requirements and timing constraints in all design phases during the development of automotive embedded systems. Both TIMMO methodology and TADL have been evaluated on prototype validators. To the best of our knowledge there is no concrete industrial implementation of TIMMO. In TIMMO-2-USE project [15], the results of TIMMO will be further validated and brought to the industry.

ProCom [16] is a two-layer component model for the development of distributed embedded systems. ProCom is inspired by RCM, and there are a number of similarities between the ProSave modeling layer (a lower layer in ProCom) and RCM. For example, components in both ProSave and RCM are passive. Similarly, both models clearly separate data flow from control flow among their components. Moreover, the communication mechanism for component interconnection used in both models is pipe-and-filter. The validation of a complete distributed embedded system, modeled with ProCom, is yet to be done. Moreover, the development environment and the tools accompanying ProCom are still evolving.

IV. Proposed Solution

A. Previous Approach

We introduced special-purpose components, i.e., Network Specification (NS), Output Software Circuit (OSWC) and Input Software Circuit (ISWC) in [2] to support modeling of real-time network communication in distributed real-time systems. This approach currently supports Controller Area Network (CAN) [17] and its high-level protocols. The NS component is the model of communication in a physical network. The protocol-independent part of NS defines messages, data-elements mapped to these messages, message properties, i.e., a message ID, a unique sender node ID, a list of receiver nodes IDs and an ordered set of signals. The protocol-dependent part of NS defines the behavior semantics of each message according to the protocol used for network communication. It contains complete information of all the frames which are sent on the bus. Moreover, it contains a Signal Mapping object that defines all the rules concerning the following questions. How are signals packed into the frames? How many signals a message contains? How are signals encoded into the frames at the sender node? How are signals decoded from the received frames and sent to the respective SWCs at the receiver node? Hence, the linking information of all distributed transactions (i.e., the transactions that are distributed over more than one node) in the modeled distributed real-time application is provided in NS.

The OSWC component is the model representation of signals in an outgoing message to the network. OSWC has only one trigger in-port and at least one data in-port. Each data in-port is associated with one signal in NS. OSWC has no data and trigger out-ports. It uses protocol-specific rules, specified in the protocol-specific part of NS, while encoding data and mapping signals to a frame. In this way, OSWC provides a clear abstraction to the SWCs that send signals to one of its data in-ports. Thus, SWCs are kept unaware of the protocol-specific details such as signal-to-frame mapping, data type encoding and transmission patterns of frames.

The ISWC component is the model representation of signals in an incoming message from the network. It has one unconditional trigger out-port that produces a trigger signal every time it is executed. There is at least one data out-port which is associated with one signal in NS. It has no data out-ports. There is one trigger in-port which is triggered when a frame arrives from the network. When a frame arrives at a node, the physical network drivers and protocol-specific implementation of ISWC extract the signals (zero or more signals per frame). When the signal (s) is delivered, the data is placed on the data port which is connected to the data in-port of the destination component (the linking information is provided in NS), and the corresponding trigger port is triggered.

In order to link the event chains, pointers (references) are assigned to the input trigger ports of all OSWCs and the output trigger ports of all ISWCs along the same distributed transaction. All such pointers for all event chains in the system are specified in NS. The model representation of OSWC, ISWC and NS in a two-node distributed real-time system is shown in Fig. 1.

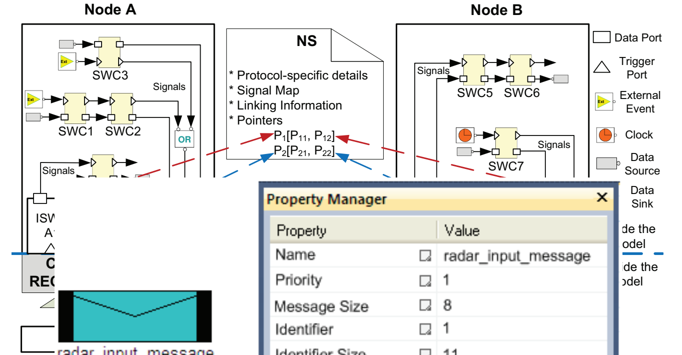


Fig. ... stem

Property		Value
Name	☐	radar_input_message
Priority	☐	1
Message Size	☐	8
Identifier	☐	1
Identifier Size	☐	11
Transmission Type	☐	Periodic
Period	☐	4 us
Inhibit Time	☐	0
Release Jitter	☐	0

T... ined prop... ifies only the name, priority, data size, value and type (in case of CAN) of identifier of the message. The message automatically inherits jitter, transmission type and period or inhibit time or both from the sender OSWC component.

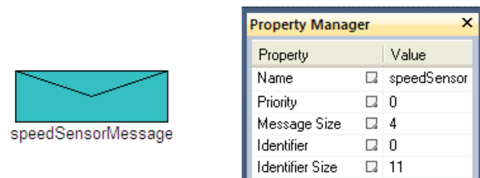


Fig. 2. Model of a message along with the list of user-defined properties

B. Proposed Extensions

One simple solution to the problem (discussed in Section II) could be using the model of a dummy sender node in place of the ECU that is not available but decisions on the messages it sends are already taken. The sole purpose of this node will be encoding and packing signals into messages and sending them to the network. Similarly, a dummy receiver node can be used to receive messages in place of a missing ECU. Such a solution can be realized with sender and receiver nodes that contain one OSWC and one ISWC for every message they send and receive respectively. However, this solution is impractical as it adds design complexity to the system. Moreover, it forces an extra modeling and architecture overhead on the developer because there are several modeling and specification steps involved when a node is modeled.

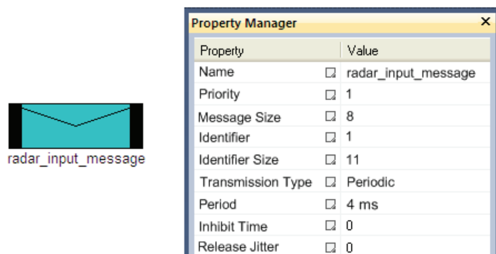
The problem can be solved in a better way by introducing to the component technology a special type of message called the stand-alone message. This message supports the modeling

[1]Developer refers to the application developer. We will overload the terms "developer" and "user" throughout the paper.

of “outside traffic”. It does not bear any association with the OSWC component. This means, it does not have any sender task inside the model of the system. However, there is an option for the user to associate this message to any number of ISWC components, i.e., it can be received by any number of tasks inside the model of the system. Apart from name, priority, data size, value and type of identifier (user-defined properties for a regular message), it is also possible for the user to specify transmission type and corresponding timing parameters (period, inhibit time or both) for this message.

The transmission type of a message is a very important parameter because the network analysis is dependent upon it especially in the case of CAN and its high-level protocols. For example, if there are only periodic and sporadic messages in the system then one type of network analysis is used [18]. On the other hand, if there is at least one mixed message in the system then another type of analysis (i.e., analysis for mixed messages) is used [10], [19], [20].

The user can also specify release jitter for the stand-alone message that may either be equal to the estimated response time of the sender task (belonging to the node that is not available) or zero (if it cannot be estimated at this stage). The extra user-defined information in the case of stand-alone messages is vital for the end-to-end timing analysis. This is because it is not possible to conduct the network analysis without availability of this timing information. The standalone message introduced in Rubus-ICE along with the list of its user-defined properties is shown in Fig. 3. The dark vertical stripes on both ends of the stand-alone message differentiates it from the regular message in RCM.



Property	Value
Name	radar_input_message
Priority	1
Message Size	8
Identifier	1
Identifier Size	11
Transmission Type	Periodic
Period	4 ms
Inhibit Time	0
Release Jitter	0

Fig. 3. Model of a stand-alone message with the list of user-defined properties

The missing timing information (that is required to perform the holistic or end-to-end timing analysis) is acquired from the user-defined properties. These messages are treated differently from the regular messages at the attribute inheritance step by the holistic response-time analysis algorithm [11], [3].

It should be noted that the list of user-defined properties of a stand-alone message (see Fig. 3) is more general and includes user-defined properties of a regular message (see Fig. 2). One may think of using the user-defined properties in Fig. 3 uniformly for all types of messages. However, this is not practical mainly because of two reasons. First, the timing information extracted from the modeled application may be redundant. That is, the transmission type and corresponding period and inhibit time will be extracted from the user-defined input as well as from the sender task. This redundancy may result in the extraction of ambiguous end-to-end timing model. Consequently, the calculated response times and delays may be erroneous. Second, it will add extra complexity and burden on the developer to specify too much information during the modeling of the system. Our intension is to extract unambiguous timing information for the end-to-end timing analysis and keep things as simple as possible for the developer.

V. Automotive Application Case Study

We provide a proof of concept for our extended method that we implemented in Rubus-ICE by conducting an automotive-application case study. We model the next-generation Adaptive Cruise Control system that contains “outside traffic” which is modeled by means of stand-alone messages. We also analyze the modeled system using the Holistic Response Time Analysis (HRTA) plug-in [4], [3] in Rubus-ICE.

A. Next-Generation Adaptive Cruise Control System

The Adaptive Cruise Control (ACC) system is an automotive feature that allows a vehicle to automatically adapt itself to the traffic environment to maintain a steady speed to the value that is preset by the driver. Often, it uses a radar to create a feedback of distance to and velocity of the preceding vehicle. It also communicates (cooperates) with the surrounding vehicles. Moreover, it receives traffic related cloud-services, i.e., community map and turn-by-turn navigation services from outside of the 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. The ACC system may be modeled with four nodes, i.e., Cruise Control (CC), Engine Control (EC), Brake Control (BC) and User Interface (UI) node [21]. Fig. 4 shows the block diagram of ACC system. The nodes communicate with each other via a CAN network.

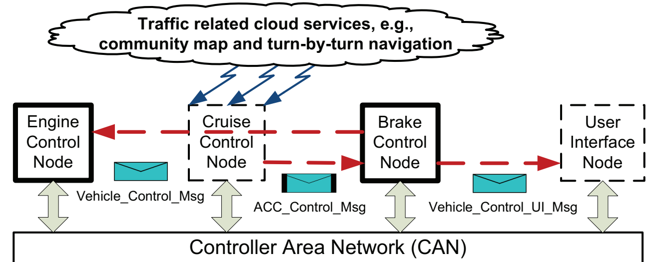


Fig. 4. Block diagram of Adaptive Cruise Control System

Assume that the models of EC and BC nodes are available while the models of CC and UI nodes will be available at a later stage. However, the decisions about network communication have been made. There is one stand-alone message “ACC_Control_Msg” in the system that is assumed to be sent by the CC node (not available yet). This message is received by the BC node as shown by the broken-line arrow in Fig. 4. Similarly, the BC node sends a message to the UI node (not available yet) and EC node. It should be noted that the broken-line arrows represent virtual communication while the actual message transmission will take place via CAN bus.

1) User Interface Node: The UI node reads driver inputs 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 states for ACC; Set Speed input (desired cruising speed set by the driver) and desired clearing distance from the preceding vehicle. This node also receives linear and angular speed, status

of manual brake sensor, and status messages and warnings to be displayed on the screen from the BC node via CAN bus.

2) Cruise Control Node: The CC node analyzes the state of the cruise control switch; if it is in the ON state then it activates the cruise control functionality. It reads input from proximity sensor (e.g., radar) and processes it to determine the presence of a vehicle in front of it. It also receives V2V communication and navigation information from outside of the vehicle as shown in Fig. 4. Moreover, it processes the radar signals along with the other information such as vehicle speed to determine its distance from the preceding vehicle. Accordingly, it sends control information as a CAN message to the BC node to adjust the speed of the vehicle with the cruising speed or clearing distance from the preceding vehicle.

3) Engine Control Node: It is responsible for controlling vehicle speed by adjusting engine throttle. It reads sensor input and accordingly determines engine torque. It receives a CAN message (from the BC node) that includes information regarding vehicle speed and status of manual brake sensor. Based on the received information, it determines whether to increase or decrease engine throttle. It then sends new throttle position to the actuators that control engine throttle.

4) Brake Control Node: The BC node 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 node. Based on this feedback, it computes new vehicle speed. Accordingly, it produces control signals and sends them to the brake actuator and brake light controller. It also sends CAN messages to EC and UI nodes that carry information regarding status of manual brake, vehicle speed and RPM.

B. Modeling of the ACC system in Rubus-ICE

The RCM model of ACC system is shown in Fig. 5. Since, CC and UI nodes are not available at this stage, there are only two nodes, i.e., CC and EC in the modeled application. The model of CAN bus is also shown. The selected speed of CAN bus is 500 kbps. The standard frame format is selected which means that all frames will use 11-bit identifier [17].

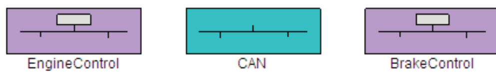
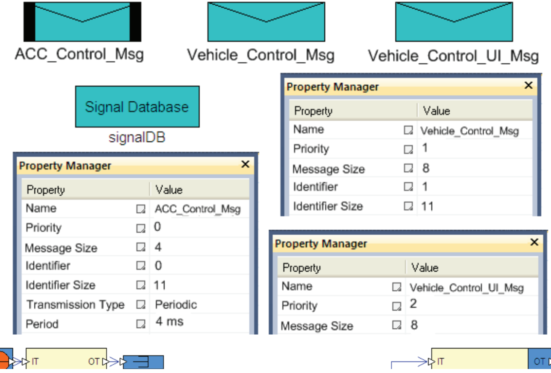
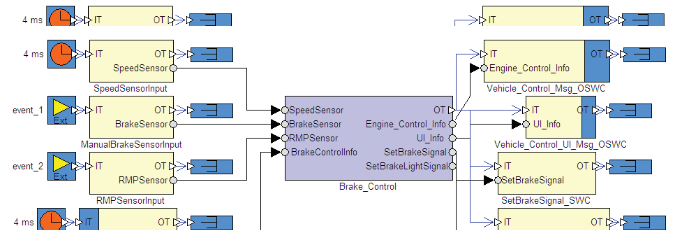


Fig. 5. Adaptive Cruise Control System modeled with RCM

There are three CAN messages in the system, i.e., ACC_control_Msg, Vehicle_Control_Msg and Vehicle_Control_UI_Msg as shown in Fig. 6. ACC_control_Msg is the only stand-alone message. The senders and receivers of all messages are shown in Fig. 4. A signal data base that corresponds to NS (see Section IV) contains all the signals sent to the network is also shown in Fig. 6. Each signal in the signal database is linked to one or more messages. The user-defined properties of all messages are also visible in Fig. 6.

The internal architecture of the BC node is shown in Fig. 7. It is modeled with five SWCs (i.e., SpeedSensorInput, ManualBrakeSensorInput, RMPSensorInput, SetBrakeSignal_SWC and SetBrakeLightSignal_SWC), one ISWC component (i.e., ACC_control_Msg_ISWC), two OSWC components (i.e., Vehicle_Control_Msg_OSWC and Vehicle_Control_UI_Msg_OSWC) and one assembly (i.e., Brake_Control). An assembly in RCM is a container for various software items. The Brake_Control assembly is further modeled with two SWCs, i.e., BrakeInputInfoProcessing and BrakeController as shown in Fig. 8. Each component is named after its functional behavior, e.g., the ACC_control_Msg_ISWC component is responsible for sending ACC_control_Msg to the network.





formationProcessing and ThrottleControl as shown in Fig. 10.

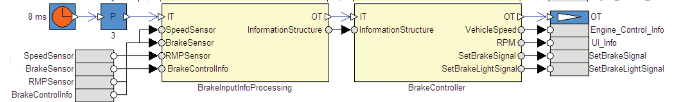


Fig. 8. Internal model of Brake_Control assembly in RCM

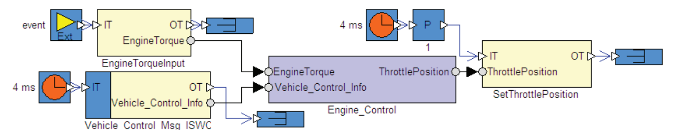


Fig. 9. RCM model of the Engine Control node

C. Holistic Response-Time Analysis of the ACC System

The HRTA plug-in in Rubus-ICE [3] is able to compute the response times of all messages and tasks as well as end-to-end or holistic response times of Distributed Transactions (DTs),

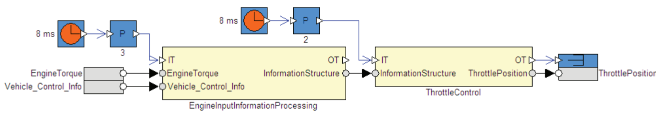


Fig. 10. Internal model of Engine_Control assembly in RCM

TABLE I
CALCULATED HOLISTIC RESPONSE TIMES OF DTS UNDER ANALYSIS

DT	DT_1	DT_2	DT_3	DT_4
HRT (us)	520	555	1250	830

i.e., task chains that are distributed over more than one node in the system. We refer the reader to [3] for the details about the holistic response-time analysis under consideration. We will focus on the analysis of the following DTs.

1) DT_1: ACC_control_Msg → ACC_control_Msg _ISWC → BrakeInputInfoProcessing → BrakeController → SetBrakeSignal_SWC
2) DT_2: ACC_control_Msg → ACC_control_Msg _ISWC → BrakeInputInfoProcessing → BrakeController → SetBrakeLightSignal_SWC
3) DT_3: SpeedSensorInput → BrakeInputInfo Processing → BrakeController → Vehicle_Control_ UI_Msg_OSWC → Vehicle_Control_UI_Msg
4) DT_4: SpeedSensorInput → BrakeInputInfo Processing → BrakeController → Vehicle_Control_ Msg_OSWC → Vehicle_Control_Msg → Vehicle_ Control_Msg_ISWC → EngineInputInformation Processing → ThrottleControl → SetThrottlePosition

Both DT_1 and DT_2 are initiated by the stand-alone message ACC_control_Msg. They terminate by producing the control signals for brake actuators and brake light controllers. DT_3 starts with the speed sensor input in the BC node and terminates by sending the message destined for UI node whose model is not available at this stage. Finally, DT_4 initiates with the speed sensor input in the BC node and terminates by producing a control signal for engine throttle controller in the EC node. The worst-case execution times of all SWCs are selected from the range of $(20 - 200)us$. The Holistic Response Times (HRTs) of these DTs are shown in Table I.

VI. CONCLUSION AND FUTURE WORK

The support for early end-to-end timing analysis of vehicular distributed real-time systems is often among industrial requirements which the tool suppliers are supposed to fulfill. The models of some subsystems or nodes may not be available at early stages during the development process and hence, the system may contain network traffic originating from outside of the system model. We extended our previous methods to support modeling and early end-to-end timing analysis of automotive distributed real-time applications that use such "outside traffic". As a proof of concept, we implemented our extended method in the existing industrial tool suite Rubus-ICE which is used for the model- and component-based development of software for vehicular embedded systems. We also conducted automotive application-case study by modeling ACC system at an early stage where the models of some nodes were not available. However, stand-alone messages were present in the system because the design decisions about network communication were already taken. We also performed end-to-end timing analysis of the the modeled ACC system.

The analysis results indicate that our extended method is sound, and it can be used for modeling and timing analysis of distributed real-time systems at early stages during the development. Moreover, our approach is equally applicable to the network traffic received from cloud-based applications. We believe, this simple yet effective approach may be suitable for other component models that use pipe-and-filter style for components interconnection, e.g., ProCom. This approach is also a step towards models inter-operability such that different component technologies may be used together for the development of complex distributed real-time systems [22].

ACKNOWLEDGEMENT

This work is supported by the Swedish Knowledge Foundation (KKS) within the project FEMMVA. We thank the industrial partners Arcticus Systems, BAE Systems Hägglunds and Volvo Construction Equipment (VCE), Sweden.

REFERENCES

[1] "Waze," http://www.waze.com/.
[2] S. Mubeen, J. Mäki-Turja, M. Sjödin, and J. Carlson, "Analyzable modeling of legacy communication in component-based distributed embedded systems," in *37th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, Sep. 2011, pp. 229–238.
[3] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "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*, April 2012, pp. 210 –221.
[4] "Arcticus Systems," http://www.arcticus-systems.com.
[5] 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.
[6] "BAE Systems Hägglunds," http://www.baesystems.com/hagglunds.
[7] "Volvo Construction Equipment," http://www.volvoce.com.
[8] "Mecel," web page, http://www.mecel.se.
[9] "Knorr-bremse," web page, http://www.knorr-bremse.com.
[10] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "Extending schedulability analysis of controller area network (CAN) for mixed (periodic/sporadic) messages," in *16th IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, sept. 2011.
[11] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocess. Microprogram.*, vol. 40, pp. 117–134, April 1994.
[12] "AUTOSAR Techincal Overview, Version 2.2.2. AUTOSAR – AUTomotive Open System ARchitecture, Release 3.1, The AUTOSAR Consortium, Aug., 2008," http://autosar.org.
[13] "TIMMO Methodology , Version 2," *TIMMO (TIMing MOdel), Deliverable 7*, October 2009, The TIMMO Consortium.
[14] "TADL: Timing Augmented Description Language, Version 2," *TIMMO (TIMing MOdel), Deliverable 6*, Oct. 2009, The TIMMO Consortium.
[15] "TIMMO-2-USE," http://www.timmo-2-use.org/.
[16] 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.
[17] ISO 11898-1, "Road Vehicles interchange of digital information controller area network (CAN) for high-speed communication, ISO Standard-11898, Nov. 1993."
[18] K. Tindell, H. Hansson, and A. Wellings, "Analysing real-time communications: controller area network (CAN)," in *Real-Time Systems Symposium (RTSS) 1994*, pp. 259 –263.
[19] S. Mubeen, J. Mäki-Turja and M. Sjödin, "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.
[20] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "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)*, sept. 2012.
[21] "Adaptive Cruise Control System Overview," in *Workshop of Software System Safety Working Group*, April 2005.
[22] S. Mubeen, M. Sjödin, J. Mäki-Turja, K.-L. Lundbäck, and P. Wallin, "Automated Model Translations for Vehicular Real-Time Embedded Systems with Preserved Semantics," in *ACM SIGBED Review: Special Issue on 33rd IEEE Real-Time Systems Symposium (WIP)*, Dec. 2012.