Extracting Timing Models from Component-based Multi-criticality Vehicular Embedded Systems

Saad Mubeen*, Mattias Gålnander†, John Lundbäck†, Kurt-Lennart Lundbäck†
*Mälardalen University, Västerås, Sweden
†Arcticus Systems AB, Järfälla, Sweden
*saad.mubeen@mdh.se; †{mattias.galnander, john.lundback, kurt.lundback}@arcticus-systems.com

***Abstract*—Timing models include crucial information that is required by the timing analysis engines to verify timing behavior of vehicular embedded systems. The extraction of this information from these systems is challenging due to the software complexity, distribution of functionality and multiple criticality levels. To meet this challenge, this paper presents a comprehensive end-to-end timing model for multi-criticality vehicular distributed embedded systems. The model is comprehensive, in the sense that it captures detailed timing information and supports various types of real-time network protocols used in the vehicular domain. Moreover, the paper provides a method to extract these models from the software architectures of these systems. The proposed model is aligned with the component models and standards in the vehicular domain that support the pipe-and-filter communication among their basic building elements.**

I. Introduction

Many vehicular embedded systems are real-time systems. This means, the times at which these systems provide their responses are as important as functional correctness of the responses. The manufacturer of such a system is required to ensure that logically correct response by the system will be provided at the time that is appropriate for the system and its environment. This is often mandated by the certification bodies in the case of safety-critical vehicular embedded systems. The appropriate time for response delivery is defined by the timing requirements that are specified on the system. Note that not all functions in modern vehicles have real-time requirements. In fact, the vehicle software consists of functions with different criticality levels, e.g., some functions are safety-critical with stringent real-time requirements, some are not safety-critical but have real-time requirements, and the rest are non-critical functions. As a result, the vehicle software has multiple criticality levels. The main challenge for the developers of these systems is to support the development of multi-criticality software in a reliable and cost-effective manner.

Model- and component-based software development has emerged as a promising approach to handle the complexity of vehicle software [1]. This approach allows to use models throughout the development process, raises the level of abstraction during the software development, supports separation of concerns, allows to build large software systems from pre-existing and reusable software components and their architectures, and supports automation. One remarkable advantage of this approach is that it allows to extract the end-to-end timing information from the software architectures and use the extracted information to populate the end-to-end timing models earlier during the development of these systems. The end-to-end timing models are vital in performing the end-to-end timing analysis [2], [3] of the systems.

There are several component models in the vehicular domain that support development of vehicular distributed embedded systems, following the model- and component-based software development approach, e.g., AUTOSAR [4], Rubus Component Model (RCM) [5], ProCom [6], COMDES [7], CORBA [8], just to name a few. The end-to-end timing model and the model extraction method presented in this paper conform to any component model that supports a pipe and filter style for communication. Hence, the presented model is aligned with the above mentioned component models.

Mixed-criticality is becoming a well-studied topic in the real-time systems community [9]. The mixed-criticality model is based on the work by Vestal [10], where a task (a run-time entity corresponding to a software component) is assumed to have more than one criticality level. In comparison, the multi-criticality model, which is part of the end-to-end timing model presented in this paper, associates a unique criticality level to the application software and not to individual components (or tasks). The presented model is inspired by the functional safety standard for road vehicles ISO26262 [11] and the aerospace standard DO178C [12]. Timing model for the AUTOSAR standard was developed in the TIMMO2USE project [13] using the TADL2 [14] language. The timing requirements model, part of the presented end-to-end timing model, is aligned with the timing constraints in TADL2. There are a few works that extract timing models from distributed embedded systems such as [15], [16]. Unlike the model presented in this paper, these models are limited to single-criticality systems.

This paper presents a comprehensive end-to-end timing model for multi-criticality distributed vehicular embedded systems. The model incorporates several real-time network protocols that are used in the vehicle industry today. The paper also presents a method for the extraction of the end-to-end timing models from the software architectures of the systems that are developed using the model- and component-based software development approach. Moreover, the paper discusses the consequences of extracting unambiguous timing model from the software architecture with a concrete example. The proposed model and method are generally applicable to any component model for distributed embedded real-time systems that supports a pipe-and-filter communication style for interaction among the software components.

II. End-to-end Timing Model in Vehicular Systems

An end-to-end timing model contains all the information that is required by analysis engines to perform the end-to-end timing analysis of a distributed embedded system. This

information includes timing properties, requirements and dependencies in the system. The relationship among the software architecture, the end-to-end timing model and the timing analysis engines is depicted in Fig 1. The end-to-end timing model consists of three models: (1) timing model, (2) linking model and (3) timing requirements model, as shown in Fig 1.

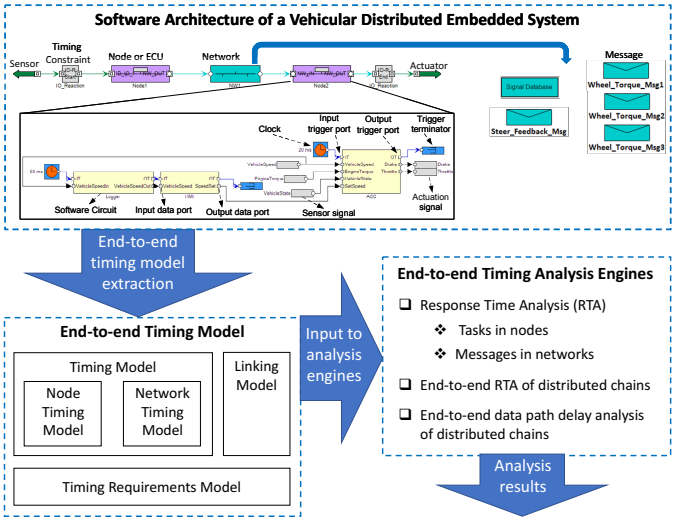

Fig. 1: Relationship among the software architecture, end-to-end timing model and timing analysis engines.

A. Timing Model

A distributed embedded system, denoted by $\mathcal{S}$, consists of two or more nodes and one or more networks. A node or an ECU can be denoted by $\mathcal{E}$, whereas a network is denoted by $\mathcal{N}$. Hence, the system can be represented as follows.

$$\mathcal{S} := \langle \{\mathcal{E}_1, \ldots, \mathcal{E}_n\}, \{\mathcal{N}_1, \ldots, \mathcal{N}_m\} \rangle \tag{1}$$

That is, the system consists of n number of nodes and m number of networks. This paper considers the node and network timing models separately. Together these two models comprise the system timing model.

*1) **Node Timing Model**:* This model contains all the timing information within a node. This model is based on the transactional task model [17], [18], [19], [16]. The most important aspect of the transactional task model is that it models tasks with offsets, externally imposed time intervals between the arrivals of the triggering events and release (for execution) of the corresponding tasks.

A node, $\mathcal{E}_i$, consists of one or more *partitions*. A partition is denoted by $\mathfrak{P}_{ij}$. The first subscript, *i*, represents the node to which this partition belongs, whereas the second subscript, *j*, represents the index of the partition within the node. The number of partitions in node $\mathcal{E}_i$ is represented by $|\mathcal{E}_i|$. The node $\mathcal{E}_i$ can be represented as follows.

$$\mathcal{E}_i := \{\mathfrak{P}_{i1}, \ldots, \mathfrak{P}_{i|\mathcal{E}_i|}\} \tag{2}$$

The partition represents the logical division of a node into multiple execution resources. The partition provides a mechanism to isolate the software within a node in both time and space. Separation in time means that each partition gets a reserved share of the nodes processing time for the execution of the software allocated to it. Separation in space means that the memory available to each node is divided among its partitions. Each partition executes a part of the system with specific criticality. Hence, a criticality level, denoted by $\mathfrak{C}_{ij}$, is associated to each partition. The criticality levels conform to the four Automotive Safety Integrity Levels (ASIL A, ASIL B, ASIL C and ASIL D) that are defined in the ISO 26262 functional safety standard for road vehicle [11]. According to the standard, ASIL D is the highest safety integrity level, whereas ASIL A is the lowest safety integrity level. Note that the presented timing model can be easily adapted to the multi-criticality aerospace embedded systems by considering the five criticality levels (A-E) specified in the DO-178C standard [12].

A partition, $\mathfrak{P}_{ij}$, consists of a set of $|\mathfrak{P}_{ij}|$ transactions. Let a transaction be denoted by Γ_{ijk}. The first and second subscripts, *i* and *j*, represent IDs of the node and partition to which this transaction belongs respectively. The third subscript, *k*, denotes the index of the transaction within the partition. Hence, the partition $\mathfrak{P}_{ij}$ can be represented by the following tuple.

$$\mathfrak{P}_{ij} := \langle \{\Gamma_{ij1}, \ldots, \Gamma_{ij|\mathcal{E}_k|}\}, \mathfrak{C}_{ij} \rangle \tag{3}$$

Each transaction Γ_{ijk} is assumed to be activated by mutually independent events with arbitrary phasing. This means, the activating events can be periodic with a periodicity of T_{ijk} or sporadic with T_{ijk} representing the minimum inter-arrival time between two consecutive events. The transaction Γ_{ijk} contains $|\Gamma_{ijk}|$ number of tasks. Each task in Γ_{ijk} may not be activated until a certain time, known as the *offset*, elapses after the arrival of the event. An offset also specifies temporal dependency among releases of tasks within the transaction.

A task is denoted by τ_{ijkl}. The first, second and third subscripts, *i*, *j* and *k*, denote the IDs of the node, partition and transaction to which the task belongs. The fourth subscript, *l*, specifies the ID the task within the transaction. A transaction belonging to partition $\mathfrak{P}_{ij}$ can be represented as follows.

$$\Gamma_{ijk} := \langle \{\tau_{ijk1}, \ldots, \tau_{ijk|\Gamma_{ijk}|}\}, T_{ijk} \rangle \tag{4}$$

A task, τ_{ijk}, is defined by the following tuple.

$$\tau_{ijkl} := \langle C_{ijkl}, T_{ijkl}, O_{ijkl}, P_{ijkl}, J_{ijkl}, B_{ijkl}, R_{ijkl}, D_{ijkl} \rangle \tag{5}$$

Where, C_{ijkl} denotes the worst-case execution time of the task. O_{ijkl} and P_{ijkl} represent the offset and priority of the task respectively. J_{ijkl} denotes the maximum release *jitter* of the task. Jitter is the difference between the earliest and the latest point in time a task starts to execute (relative to its nominal start time). B_{ijkl} denotes the maximum blocking time for the task. B_{ijkl} is defined as the maximum amount of time the task has to wait for a shared resource that is already locked by a lower priority task. The upper bound on the blocking time for a task can be obtained by using a resource sharing (synchronization) protocol such as Stack Resource Policy (SRP) [20] and Priority Ceiling Protocol (PCP) [21]. R_{ijkl} represents the worst-case response time of the task. D_{ij} denotes the deadline of the task. In this model, there are no restrictions on the offset, deadline or jitter. This means that each of these parameters can each be smaller that, equal to or greater than the task corresponding period.

*2) **Network Timing Model**:* This model contains all the timing information within a network. The model considers various real-time network protocols that are used in the vehicular domain. These networks include broadcast protocols such as Controller Area Network (CAN) [22], CANopen [23], HCAN [24] AUTOSAR COMM [25] as well as the point-to-point communication protocols based on switched Ethernet such as AVB [26] and HaRTES [27]. A network, $\mathcal{N}_i$, is defined by the following tuple.

$$\mathcal{N}_i := \langle \mathfrak{Z}_i, \mathfrak{S}_i, \mathcal{L}_i, \mathfrak{T}_i, \mathcal{W}_i, \mathcal{M}_i \rangle \tag{6}$$

Where $\mathfrak{Z}_i$ represents the speed of the network, often represented in Kbit/s or Mbit/s. $\mathfrak{S}_i$ represents the set of switches in the case of a multi-hop network, for example Ethernet AVB. In a multi-hop network, a switch is connected to a node or to another switch by a link. All such links in the network are represented by the set $\mathcal{L}_i$. In the Ethernet AVB protocol and the Time Sensitive Network (TSN) protocol [28], different types of traffic can be specified a portion of the total bandwidth by means of slopes. For example the Ethernet AVB protocol specifies slopes for real-time traffic (Class A with higher priority and Class B with lower priority) and non real-time traffic. The slopes of different types of traffic are represented in the set $\mathfrak{T}_i$. In some switched Ethernet protocols like HaRTES, the transmission is performed within pre-configured fixed-duration time slot called Elementary Cycle (EC). The EC consists of two windows that are dedicated to synchronous and asynchronous traffic. The size of the EC and the two windows is represented in the set $\mathcal{W}_i$. Finally, $\mathcal{M}_i$ denotes the set of messages that are communicated over the network. $\mathcal{M}_i$ is represented by the following tuple.

$$\mathcal{M}_i := \langle \mathcal{X}_{ij}, \mathcal{F}_{ij}, P_{ij}, C_{ij}, s_{ij}, T^P_{ij}, T^S_{ij}, J_{ij}, O_{ij}, B_{ij}, R_{ij} \rangle \tag{7}$$

There are two subscripts associated to each term in the above tuple. The first subscript, *i*, represents the network to which the message set belongs. The second subscript, *j*, specifies a unique identifier for each message in the message set. $\mathcal{X}_{ij}$ specifies the type of the message. A message can be periodic, sporadic or mixed (both periodic and sporadic) [29]. $\mathcal{F}_{ij}$ specifies the frame type, i.e., whether the frame is a Standard or an Extended frame in the case of CAN and its higher-level protocols. P_{ij} denotes the message priority. C_{ij} represents the worst-case transmission time of the message (considering no interferences). s_{ij} denotes the data payload in the message. If the transmission type of the message is periodic, T^P_{ij} denotes the period of the message. If the transmission type of the message is sporadic, T^S_{ij} represents the minimum time that should elapse between the transmission of any two consecutive instances of the message. Whereas, both T^P_{ij} and T^S_{ij} are specified in the case of a mixed message. J_{ij} and O_{ij} denote the release jitter and offset of the message respectively. B_{ij} represents the maximum amount of time during which the message can be blocked by the lower priority messages. R_{ij} represents the worst-case response time of the message.

B. Linking Model

Vehicular embedded systems are often modeled with chains of tasks and messages. A chain has one initiator and one terminator. Different chains may have the same initiator or the same terminator. A chain may reside on one node, in which case it is composed of only a sequence tasks. A chain may also be a distributed chain, in which case it is composed of a sequence of tasks and messages. An example of a distributed chain is a chain that is initiated at a sensor and terminated at an actuator, while it spans over several nodes.

A task in the chain may be activated by an independent trigger source or by its predecessor task. Moreover, a task in the chain may receive activation trigger, data or both from its predecessor task. Any two neighboring tasks in the chain may reside on the same node or two different nodes.

A message in the chain may be triggered for transmission by the predecessor task (also called sending task) in the case of passive networks like CAN or Ethernet AVB. Whereas, in the case of active networks like the HaRTES protocol, a message in the chain can be triggered for transmission by the network itself (regardless of the sending task). In the case of multi-hop networks, a message in the chain may traverse through several links and switches between the predecessor and successor tasks in the chain.

All this information regarding activations, trigger flows, data flows, mapping and linking within the chains constitute the system linking model. This information is crucial for the analysis engines to perform the end-to-end timing analysis.

C. Timing Requirements Model

Timing requirements in vehicular embedded systems are specified by means of timing constraints. The timing requirements model includes information regarding the specified constraints on all chains in the system. Note that the timing requirements on individual tasks and messages (e.g., deadlines of tasks and messages) are not part of this model as they are already included in the node and network timing models respectively. The set of all specified timing requirements in the system is denoted by $\mathfrak{R}$. The set $\mathfrak{R}$ contains *n* number of timing requirements as represented below.

$$\mathfrak{R} := \langle \{ \mathfrak{R}_1, \ldots, \mathfrak{R}_n \} \rangle \tag{8}$$

Each timing requirement $\mathfrak{R}_i$ has three attributes: (1) $Type$, (2) minimum value of the constraint, denoted by MIN, and (3) maximum value of the constraint, denoted by MAX.

$$\mathfrak{R}_i := \{ Type, MIN, MAX \} \tag{9}$$

There are eighteen timing constraints that are included in the AUTOSAR standard [4]. However, most of these constraints are specific to single or pair of events. That is, they are specific to individual tasks or a set of independent tasks that are not part of the same chain. In this model, we consider only four of these constraints, which are applicable to the chains. Hence, $\mathfrak{R}_i(Type)$ can be one of the four constraints that are defined in the timing model of the AUTOSAR standard.

$$\mathfrak{R}_i(Type) := \{ Age, Reaction, OutputSynchronization, InputSynchronization \} \tag{10}$$

The Age constraint constrains the maximum age of the data from the input to the output of the chain. The $Reaction$ constraint constrains the first output (reaction) of the chain corresponding to the data at the input of the chain, considering

the new data "just missed" the read access at the initiator element of the chain. If two chains have the same initiator but different terminators, the $OutputSynchronization$ constraint restricts the closeness of the occurrences of outputs of the two chains. In other words, this constraint defines how far apart the outputs of the two chains can occur corresponding to the same input of the chains. On the other hand, if two chains have the same terminator but different initiators, the $InputSynchronization$ constraint restricts the closeness of the occurrences of inputs of the two chains. That is, this constraint defines how far apart the inputs of the two chains can occur corresponding to the same output of the chains.

III. End-to-end Timing Model Extraction Method

In a model- and component-based software development process, the end-to-end timing model is extracted from the software architecture of the modeled system as shown in Fig. 1. There are two types of information that are extracted in the end-to-end timing model. The first type of information is explicitly specified by the user on the software architecture. This information is rather easy to extract from the properties of structural elements in the software architecture. The second type of information is not explicitly provided by the user. This information has to be extracted from the software architecture. If some of this information cannot be extracted unambiguously then assumptions are made about the missing information for the purpose of providing the complete end-to-end timing model to the analysis engines.

A. Extracting the Node Timing Model

Majority of the information in the node timing model falls into the first type of timing information (user-defined). For instance, most of the information in relations (2), (3), (4) and (5) is extracted from the user-defined properties of corresponding structural elements in the software architecture.

The criticality levels associated to partitions in relation (3) are not user-specified. The user can only specify a unique criticality level on each part of the complete software architecture, called the application. The application can reside on one or more partitions in the node. Since the main purpose of the partition element is to provide separation in time space, a partition is not allowed to host multiple applications with different criticality levels. Note that any inter-partition interference is prevented by using memory protection mechanisms. A node can host multiple applications with different criticality levels. Hence, the criticality level of a partition is extracted from the criticality level of the hosted application.

The transaction period in relation (4) is not user-specified. The user can only specify periods (or minimum inter-arrival times) of individual software components by means of clocks (or events or interrupts) as shown in Fig. 2. The corresponding task inherits the period or minimum inter-arrival time (depicted in relation (5)) from the software component. Each event- or interrupt-triggered task forms a transaction of its own. The transaction inherits the minimum inter-arrival time from the task. In the case of clock-triggered tasks, the transaction period is derived by calculating the least common multiple of the extracted periods of all tasks in the transaction (chain).

The individual deadlines of the tasks in relation (5) are not user-specified. In the case of time-triggered software component with no explicit release jitter, we assume implicit deadlines, i.e., the deadline of each task is equal to its period. Otherwise, the corresponding jitter and trigger information is sent to the analysis engines to make appropriate assumptions about the missing information.

B. Extracting the Network Timing Model

All network-level timing information shown in relation (6) and some of message timing information (including frame type, priority, offset and data payload) shown in relation (7) are user-specified. This information is extracted from the properties of the corresponding structural elements in the network and message models of the software architecture. Whereas, the worst-case transmission time, message priority, message type, period or minimum inter-transmission time, release jitter and message blocking time are not explicitly specified by the user; these properties are extracted from the software architecture.

The worst-case transmission time of a message is calculated using the data payload and network speed. In the case of CAN and its higher-level protocols, the priority of a message is extracted from its ID. Unlike the other protocols, the priority of a message is unique in CAN. Whereas, in the case of the other protocols, the priority of a message is a user-defined attribute, which is extracted from the message model in the software architecture. The blocking time of message is derived by considering the maximum value among the worst-case transmission times of all lower priority messages. Whereas, the release jitter of a message is derived by subtracting the sender task's (sender software component's) best-case response time from the worst-case response time.

The information regarding the message type is extracted from the sender software component. If the sender is triggered by a periodic clock, the message type becomes periodic. If the sender is triggered by a sporadic event or an interrupt, the message type becomes sporadic. Whereas, if the sender is triggered by both periodic clock and a sporadic event or an interrupt, the message type becomes mixed. Depending upon the message type, the message inherits the period, minimum transmission time or both from the sender. This information is crucial for the analysis engines as different analysis profiles are used to analysis different message types [29].

C. Extracting the Linking Model

The linking information for all distributed chains in the software architecture are extracted in the system linking model. Consider an example of the software architecture of a two-node multi-criticality vehicular distributed embedded system depicted in Fig. 2. There are three distributed chains in the system as follows.

$Chain_1$: SWC1 $\rightarrow$ SWC2 $\rightarrow$ Msg_1 $\rightarrow$ SWC6 $\rightarrow$ SWC5
$Chain_2$: SWC7 $\rightarrow$ Msg_2 $\rightarrow$ SWC3
$Chain_3$: SWC9 $\rightarrow$ Msg_3 $\rightarrow$ SWC3

The linking information for each chain is captured in a reference set. This set contains references to the data and trigger ports of each software component along the distributed chain. The ordering of references within this set corresponds to the ordering of software components and messages within the distributed chain. For example, the reference set for $Chain_2$

includes references to the trigger and data input and input ports of SWC7 and SWC8 together with the reference to Msg_2.

In order to extract control flows along the chains, each task τ_{ijk1} is assigned a trigger dependency attribute, denoted by $\mathfrak{TD}_{\tau_{ijk1}}$. The domain of this attribute is defined as follows.

$$\mathfrak{TD}_{\tau_{ijk1}} := \{Independent, Dependent\}$$

Where, $\mathfrak{TD}_{\tau_{ijk1}}$ is assigned *Independent* if the software component corresponding to task τ_{ijk1} is activated by an independent triggering source, e.g, SWC1, SWC3, SWC4, SWC7, SWC8 and SWC9 in Fig. 2. Whereas, $\mathfrak{TD}_{\tau_{ijk1}}$ is assigned *Dependent* if the software component corresponding to task τ_{ijk1} is activated by its predecessor software component, e.g, SWC2, SWC5 and SWC6 in Fig. 2. A precedence constraint is implicitly included between the two tasks in the case of *Dependent* triggering. This constraint restricts the activation of successor task before the completion of the predecessor task. Note that SWC2 and SWC 5 are triggered by their predecessors within the same partitions, whereas SWC6 is triggered by its predecessor via the network.

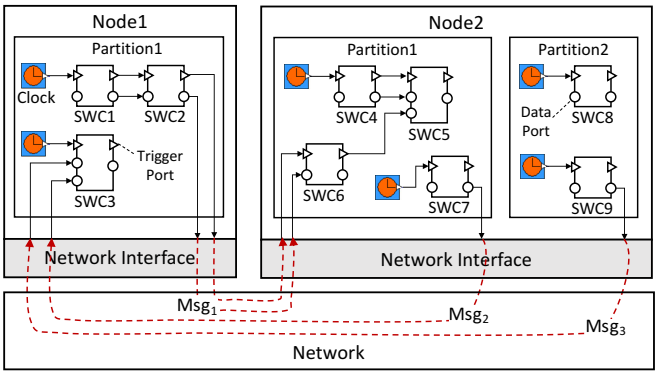

Fig. 2: A two-node multi-criticality vehicular system.

In CAN and its higher-level protocols, a message can traverse through only one link (i.e., the bus or network) between the sender and receiver software components. However, in the multi-hop switched Ethernet protocols (e.g., AVB, HaRTES), a message may traverse through several links between the sender and receiver software components. These links are extracted for every message in a dedicated set as follows.

$$\mathcal{L}_{\mathcal{M}_{ij}} := \langle \mathcal{L}_1, \ldots, \mathcal{L}_k \rangle$$

Where, $\mathcal{L}_{\mathcal{M}_{ij}}$ represents the set of k links through which the message $\mathcal{M}_{ij}$ traverses between the sender and receiver software components. In the case of CAN and its higher-level protocols, this set is equal to one link.

D. Extracting the Timing Requirements Model

The information in the timing requirements model represented with relations (8), (9) and (10) is user-defined. The user specifies the timing requirements on the chains within the software architecture by means of the "start" and "end" objects for each constraint. This means, the timing constraints can be specified on a complete chain or part of the chain. The *MAX* value of each constraint extracted from the corresponding property of the "end" object. If *MIN* value of the constraint is not specified, it is extracted in the model as zero.

IV. Discussion

It is important that the extracted information in the end-to-end timing model should be unambiguous, otherwise the timing analysis results can be under- or over-estimated. Under-estimated analysis results can have severe (or even catastrophic) consequences. For example, analysis results verify that the task responsible for the deployment of airbag will meet its deadline in the case of a crash. If the analysis results are under-estimated then the task can actually miss its deadline and may lead to catastrophe. Whereas, over-estimated analysis results can lead to underutilization of the system resources. In order to elaborate on this, consider an example of a system consisting of one distributed transaction (chain of tasks and messages) corresponding to the software architecture of a distributed chain (chain of software components and messages) as shown in Fig. 3. The chain is distributed over two nodes connected by a CAN network. There are three tasks (one in Node1 and two in Node2) and one message in the chain. The timing information about the chain is also depicted in Fig. 3. The calculations for the *Age* and *Reaction* delays depend upon several parameters including the trigger dependency attribute. This attribute for the receiving task τ_2, denoted by $\mathfrak{TD}_{\tau_2}$, can be *Independent* or *Dependent*, meaning that the receiving task implements the interrupt-based or polling-based policy to receive the message, respectively. The *Age* and *Reaction* delays for the system in Fig. 3 are shown in Fig. 4 and Fig. 5 if the receiving task implements the interrupt-based policy or the polling-based policy for receiving Msg_1, respectively. By comparing the values of the *Age* and *Reaction* delays in Fig. 4 and Fig. 5, it is clear that if the value of $\mathfrak{TD}_{\tau_2}$ is not extracted or wrongly extracted then the analysis results can be either under-estimated or over-estimated.

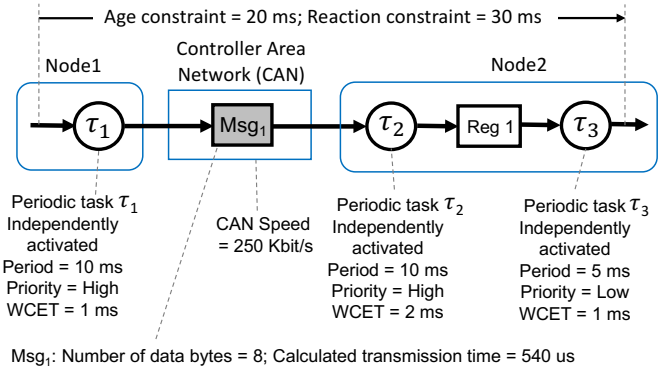

Fig. 3: Example of a distributed transaction corresponding to the software architecture of a distributed chain.

V. Summary and Conclusion

This paper has presented a comprehensive end-to-end timing model for multi-criticality vehicular distributed embedded systems. Such a model is required by the end-to-end timing analysis engines for pre-runtime verification of timing behavior of the systems (i.e., verifying the specified timing requirements before running the system). The paper takes the leverage of the principles of model- and component-based software development in providing a method to extract the end-to-end

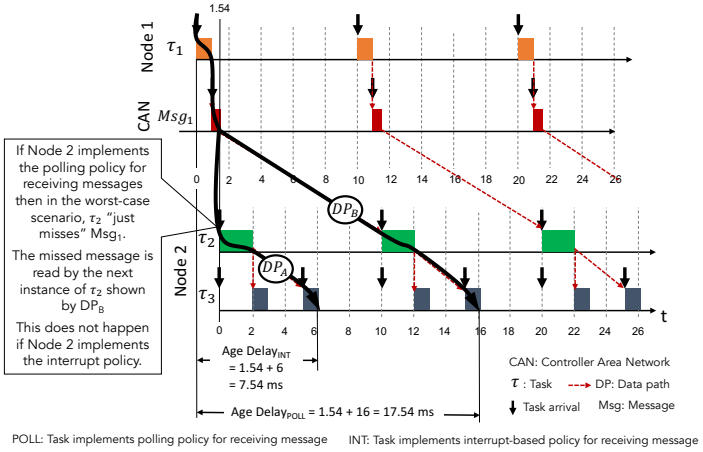

Fig. 4: Different *Age* delays in the system in Fig. 3 if Node2 uses the interrupt or polling policy for receiving messages.

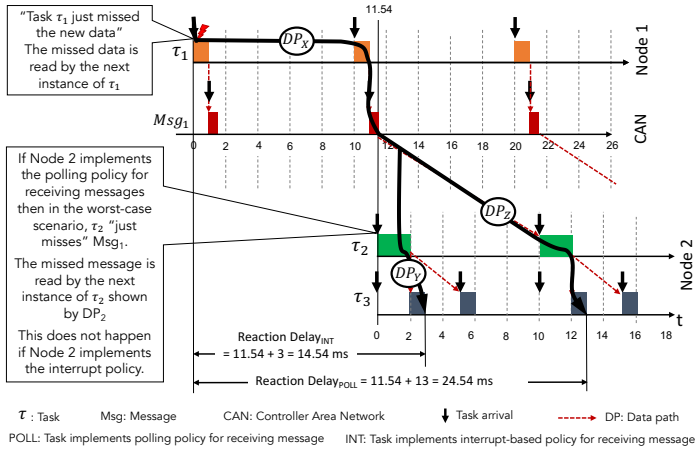

Fig. 5: Different *Reaction* delays in the system in Fig. 3 if Node2 uses interrupt or polling policy for receiving messages.

timing models from software architectures of these systems. The proposed model conforms to the component models and standards in the vehicular domain that support a pipe-and-filter communication among their software components. In the future, we plan to transfer the results to the industry by implementing the proposed model and method in the analysis framework of an existing industrial tool suite (Rubus-ICE).

Acknowledgment

The work in this paper is supported by the KKS foundation through the project PreVeiw. We thank our industrial partners Arcticus Systems, Volvo CE and BAE Systems Hägglunds.

References

[1] I. Crnkovic and M. Larsson, *Building Reliable Component-Based Software Systems*. Norwood, MA, USA: Artech House, Inc., 2002.
[2] 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.
[3] 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 *CRTS Workshop*, dec. 2008.
[4] "AUTOSAR Techincal Overview, Release 4.1, Rev.2, Ver.1.1.0." http://autosar.org.
[5] K. Hänninen et al., "The Rubus Component Model for Resource Constrained Real-Time Systems," in *IEEE Symposium on Industrial Embedded Systems*, 2008.
[6] S. Sentilles, A. Vulgarakis, T. Bures, J. Carlson, and I. Crnkovic, "A Component Model for Control-Intensive Distributed Embedded Systems," in *CBSE*, Oct. 2008.
[7] X. Ke, K. Sierszecki, and C. Angelov, "COMDES-II: A Component-Based Framework for Generative Development of Distributed Real-Time Control Systems," in *Embedded and Real-Time Computing Systems and Applications, RTCSA 2007. 13th IEEE International Conference on*, August 2007, pp. 199 –208.
[8] D. Schmidt and F. Kuhns, "An overview of the Real-Time CORBA specification," *Computer*, vol. 33, no. 6, pp. 56 –63, Jun. 2000.
[9] A. Burns and R. Davis, "Mixed criticality systems - a review, ninth edition," Dept. of Computer Science, University of York, Tech. Rep., 2017, https://www-users.cs.york.ac.uk/burns/review.pdf.
[10] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *28th IEEE International Symposium on Real-Time Systems*, Dec 2007, pp. 239–243.
[11] "International Organization for Standardization (ISO), ISO 26262-1:2011: Road vehicles – Functional safety. http://www.iso.org/."
[12] Special C. of RTCA. DO-178C, software considerations in airborne systems and equipment certification, 2011.
[13] "TIMMO-2-USE," https://itea3.org/project/timmo-2-use.html.
[14] Timing Augmented Description Language (TADL2) syntax, semantics, metamodel Ver. 2, Deliverable 11, Aug. 2012.
[15] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "Extraction of end-to-end timing model from component-based distributed real-time embedded systems," in *Time Analysis and Model-Based Design, from Functional Models to Distributed Deployments (TiMoBD) workshop located at Embedded Systems Week*. Springer, October 2011, pp. 1–6.
[16] 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.
[17] K. Tindell, "Adding Time-Offsets to Schedulability Analysis," Department of Computer Science, University of York, England, Tech. Rep., January 1994.
[18] J. Palencia and M. G. Harbour, "Schedulability Analysis for Tasks with Static and Dynamic Offsets," *Real-Time Systems Symposium, IEEE International*, p. 26, 1998.
[19] 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.
[20] T. P. Baker, "Stack-based scheduling for realtime processes," *Real-Time Systems*, vol. 3, no. 1, pp. 67–99, Apr. 1991. [Online]. Available: http://dx.doi.org/10.1007/BF00365393
[21] 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.
[22] ISO 11898-1, "Road Vehicles ? interchange of digital information ? controller area network (CAN) for high-speed communication, ISO Standard-11898, Nov. 1993."
[23] "CANopen Application Layer and Communication Profile. CiA Draft Standard 301. Version 4.02. February 13, 2002," http://www.can-cia.org/index.php?id=440.
[24] "Hägglunds Controller Area Network (HCAN), Network Implementation Specification," *BAE Systems Hägglunds, Sweden (internal document)*, April 2009.
[25] "Requirements on Communication, Rel. 4.1, Rev. 3, Ver. 3.3.1, March, 2014," www.autosar.org/download/R4.1/AUTOSAR_SRS_COM.pdf, accessed on May 05, 2014.
[26] "Audio/video bridging task group of ieee 802.1, available at http://www.ieee802.org/1/pages/avbridges.html."
[27] R. Santos, M. Behnam, T. Nolte, P. Pedreiras, and L. Almeida, "Multi-level hierarchical scheduling in ethernet switches," in *2011 Proceedings of the Ninth ACM International Conference on Embedded Software (EMSOFT)*, Oct 2011, pp. 185–194.
[28] Time-Sensitive Networking Task Group, IEEE Std 802.1Qbv-2015 - IEEE Standard for Local and Metropolitan Area Networks – Bridges and Bridged Networks, 2015.
[29] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "Integrating Mixed Transmission and Practical Limitations with the Worst-Case Response-Time Analysis for Controller Area Network," *Journal of Systems and Software*, vol. 99, no. 0, pp. 66 – 84, 2015.