2023 IEEE 26th International Symposium on Real-Time Distributed Computing (ISORC) | 979-8-3503-3902-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/ISORC58943.2023.00025

End-to-end Timing Modeling and Analysis of TSN in Component-Based Vehicular Software

Bahar Houtan*, Mehmet Onur Aybek†, Mohammad Ashjaei*, Masoud Daneshtalab*, Mikael Sjödin*, John Lundbäck†, Saad Mubeen*
*Mälardalen University, Västerås, Sweden; †Arcticus Systems, Järfälla, Sweden
*firstname.lastname@mdu.se; †firstname.lastname@arcticus-systems.com

***Abstract*—In this paper, we present an end-to-end timing model to capture timing information from software architectures of distributed embedded systems that use network communication based on the Time-Sensitive Networking (TSN) standards. Such a model is required as an input to perform end-to-end timing analysis of these systems. Furthermore, we present a methodology that aims at automated extraction of instances of the end-to-end timing model from component-based software architectures of the systems and the TSN network configurations. As a proof of concept, we implement the proposed end-to-end timing model and the extraction methodology in the Rubus Component Model (RCM) and its tool chain Rubus-ICE that are used in the vehicle industry. We demonstrate the usability of the proposed model and methodology by modeling a vehicular industrial use case and performing its timing analysis.**

I. Introduction

Designing highly software-intensive vehicular embedded systems is challenging due to the enormous size and complexity of the software [1]. In addition, the complexity in modern vehicular software is continuously growing by integration of network communication mechanisms introduced by Time-Sensitive Networking (TSN) standards [2], [3], [4]. TSN standards are attractive solutions to address the high-bandwidth and real-time communication requirements in vehicular applications. These standards enhance switched Ethernet with deterministic traffic shaping mechanisms. More specifically, TSN supports hard real-time, high bandwidth with low-latency and low-jitter traffic transmission. These features are supported by offline scheduled time-triggered traffic enabled by the Time-Aware Shaper (TAS), resource reservation for different classes of traffic by a Credit-Based Shaper (CBS), and clock synchronization [5]. Although TSN standards provide flexible design of complex and deterministic network communication, this flexibility comes with the cost of further complicating the design process due to including several configurable factors for network devices and traffic.

Model-based Engineering (MBE) and Component-based Software Engineering (CBSE) approaches [6], [7] are widely being used to manage the software complexity and for cost-effective development of vehicular software systems. There are several domain-specific languages and component models that can be used to model software architectures of vehicular systems, e.g., AUTomotive Open System ARchitecture (AUTOSAR) [8], Rubus Component Model (RCM) [9], AMALTHEA[1][10], EAST-ADL [11], and Architecture Analysis & Design Language (AADL) [12], to name a few. All the models and languages assist the embedded software developers by structuring necessary information to develop and design complex distributed embedded software systems.

In addition to managing the design complexity of vehicular distributed software systems, the developers of these systems are required to verify real-time requirements that are specified on their software architectures. The timing requirements can be verified by performing the end-to-end data-propagation delay analysis of the software architectures of these systems [13], [14]. The analysis results can also guide in the refinement of the software architectures [15]. This analysis is already implemented in several tools in the vehicle industry that support model- and component-based development of these systems, e.g., SymTA/S[2][16] and Rubus [17]. The analysis requires the end-to-end timing model as a crucial input. The end-to-end timing model includes comprehensive timing information of the vehicular distributed software system. The instances of this timing model must be extracted from the software architectures of these systems and provided as an input to the model-based analysis tools.

In this paper, we present an end-to-end timing model to represent a distributed embedded system with TSN support. Moreover, we present an automated methodology to systematically extract instances of the end-to-end timing model from the software architectures of these systems. The end-to-end timing model can be used for end-to-end data-propagation delay analysis of the systems. We evaluate the proposed methodology on an industrial use-case from the vehicular domain based on RCM developed in Rubus-ICE tool suite [18]. As a proof of concept, we chose RCM because it is the first commercially available model that supports comprehensive modeling and end-to-end data-propagation delay analysis of TSN-based vehicular distributed embedded systems [19].
The **main contributions** in this paper are as follows.

- We propose an end-to-end timing model to describe TSN networks with all configuration parameters in the distributed vehicular embedded software systems.
- We provide an automated methodology to extract instances of the end-to-end timing model from component-based software architectures of vehicular systems.

[1]https://itea4.org/project/amalthea.html
[2]SymTA/S tool has been acquired by Luxoft (https://www.luxoft.com)

979-8-3503-3902-4/23/$31.00 ©2023 IEEE
DOI 10.1109/ISORC58943.2023.00025

- We provide a proof of concept for the proposed timing model and extraction methodology by integrating them with the component-based software engineering environment of an industrial tool suite, namely Rubus-ICE.
- We evaluate the proposed model and methodology as well as their integration with the Rubus-ICE tool suite on a vehicular industrial use-case.

II. Background and Related Works

As described in [5], model-based software development of vehicular embedded systems generically consists of four steps: (1) modeling the software architecture; (2) extracting timing properties and requirements from the software architecture; (3) validate the timing requirements and constraints using end-to-end data-propagation delay analysis; and (4) refining the software architecture according to the timing analysis results.

A. Modeling the Software Architecture

There are several software architecture modeling languages and component models, such as AUTOSAR, RCM, AMALTHEA, EAST-ADL, AADL, to mention a few.

AUTOSAR is widely used for developing software architecture for automotive systems. SymTA/S is a commercial timing analysis and optimization framework that complies with AUTOSAR. A recent work in [20] integrates AUTOSAR adaptive with applicable standards to develop more sophisticated systems. The proposed three-layer architecture coordinates a binding between AUTOSAR Adaptive, OPC UA standards[3], and TSN standards. The paper argues that the proposed architecture lacks maturity since all the involved technologies in the architecture layers are still under development.

RCM comprises of hierarchical entities which are necessary to model a distributed embedded system that supports TSN. At the highest level, the system contains at least two nodes and a network element that interconnects the nodes. A node (end-station) is a processing element that provides run-time environment for one or more Software Applications (SA). The SA provides spacial and temporal isolation to the part of overall software architecture within the system. In RCM, a software architecture is modeled by interconnecting a set of Software Components (SWCs). An SWC is a design-time entity that corresponds to a task at run-time or in the timing model. The SWCs communicate with each other by their interfaces (a set of data and trigger ports).

To the best of our knowledge, RCM is the first and only component model that supports comprehensive modeling of TSN [19]. Recently, there have been some efforts in increasing the performance and applicability of the other modeling approaches to RCM by model transformation [21]. The work in [22] proposes a mapping technique between AMALTHEA and RCM to enable timing analysis of AMALTHEA-based component models in RCM. In addition, the work in [23] presents a mapping from EAST-ADL models to RCM with the aim of enabling the timing analysis of a non-RCM model.

[3]https://opcfoundation.org/about/opc-technologies/opc-ua/

B. Timing Information Extraction from Software Architectures

Modeling and extraction of timing models from the software architectures of component-based vehicular embedded systems that are developed with RCM are presented in [24]. The work in [24] considers several onboard communication protocols like Controller Area Network (CAN) [25], CANopen [26], AUTOSAR COMM [27] and switched Ethernet. However, it does not consider TSN, which is the main focus of our work.

The work in [19] complements [24] by integrating several aspects of modeling TSN standards in a component model with timing analysis perspective. However, the presented TSN timing model in [19] and the timing model extraction in [24] are not expressive enough to include all the timing aspects of TSN. Nevertheless, the complexity of TSN requires significant automation and optimization in the timing information extraction and configuration process prior to the end-to-end data-propagation delay analysis.

A recent work [28] discusses preliminary ideas and work-in-progress on extraction of timing models from TSN-based software architectures. In comparison, we propose a comprehensive end-to-end timing model and an automated methodology for the extraction of end-to-end timing models from the software architectures of vehicular distributed embedded systems, which is complemented by an automotive application case study. Moreover, our work takes into account the integration and configuration aspects of the TSN timing models.

C. End-to-end Timing Models, Timing Analysis and Verification

In order to verify the timing requirements of the chains of tasks (SWC at the design time) and network messages in the distributed embedded systems, end-to-end data-propagation delays (data age and reaction time delays) are calculated and compared with the corresponding data age and reaction time constraints. The timing constraints corresponding to the data age and reaction time delays are part of the timing model of AUTOSAR standard. The data age delay is the time elapsed between the arrival of data at the input of the chain and the latest availability of the corresponding data at the output. Whereas, the reaction time delay corresponds to the earliest availability of the data at the output of the chain corresponding to the data that just missed the read access (of the event) at the input of the chain [29], [13].

The data age and reaction time delays in a two-node distributed embedded system which is modeled based on the conventional task model and with one TSN message are demonstrated in Fig. 1. The data flows between the task instances and messages are indicated by dashed orange arrows in Fig. 1 to indicate the read and write of data in the chain.

There are several works that have also presented end-to-end data-propagation delay analysis based on the conventional task model [14], [30]. The work in [17], [15] considered the analysis for distributed embedded systems that are based on CAN network. The work in [31] considers the systems that use Ethernet networks. The works in [14] and [32] presented the end-to-end data-propagation delay analysis for vehicular applications, where some of the techniques have been

implemented in tools to support component-based software development, e.g., [17] and [33]. The works in [34], [18] present open research challenges and their solutions when integrating the timing analysis with model-based development tools. However, these works only focus on traditional onboard networks such as CAN without considering TSN.

In the case of TSN, there are several additional features that need to be considered such as synchronization of the end-stations, and the presence of different shaping mechanisms for different TSN classes. A recent work in [35] shows that the existing end-to-end data-propagation delay analysis supports the non-scheduled TSN classes, but it does not support the scheduled traffic in TSN.

The end-to-end data-propagation delay analysis results are typically presented in the form of tables showing individual response time of each task (SWC) or message and the data age and reaction time delays. Such information can be used by the system designer and integrator to refine the software architecture [15]. Automated refinement of the software architecture according to the analysis results is an open research challenge [5] that could be beneficial for improving the scalability and efficiency of the model-based software development for more sophisticated vehicular embedded systems.

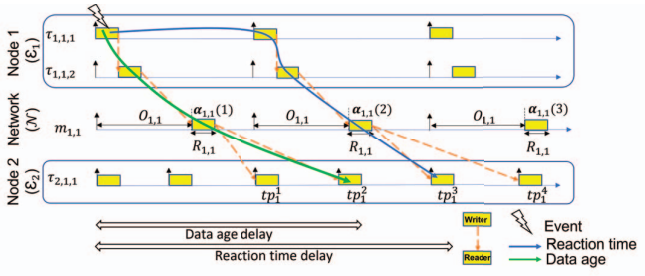

Fig. 1: Demonstration of data age and reaction time delays.

III. End-to-end Timing Model

In this paper, we consider the end-to-end timing model that captures the timing information from the software architecture of TSN in the distributed embedded systems. Such a model is a necessary input for the analysis techniques and tools to verify the timing behavior of the software architecture. The system model consists of two or more end-stations (or nodes), denoted by $\mathcal{E}$, and at least one TSN network, denoted by $\mathcal{N}$ as shown in Eq. (1):

$$\mathcal{S} := \langle \mathcal{E}, \mathcal{N} \rangle \tag{1}$$

The set of networks in the system model is defined by Eq. (2) as follows:

$$\mathcal{N} := \{\mathcal{N}_1, ..., \mathcal{N}_{|\mathcal{N}|}\} \tag{2}$$

The networks connect end-stations to each other. The overall set of end-stations in the system model is denoted by $\mathcal{E}$, as shown in Eq. (2):

$$\mathcal{E} := \{\mathcal{E}_1, ..., \mathcal{E}_{|\mathcal{E}|}\} \tag{3}$$

Various components of the end-to-end timing are discussed in the following subsections.

A. End-station Timing Model

The system model considers single-core end-stations similar to the model in [19]. In such a system, the computation and storage resources need to be shared between the SWCs in an isolated manner. A set of SWCs contributing to a software functionality are isolated in an application within the end-station.

The end-station consists of one or more applications as shown in Eq. (4). The application provides spacial and temporal isolation of the software functionality. In other words, the applications are isolated in terms of allocated time for execution, and also allocated memory.

$$\mathcal{E}_i := \{A_{i1}, ..., A_{i|\mathcal{E}_i|}\} \tag{4}$$

Above, i indicates the end-station ID that the application belongs to. $|E_i|$ is the total number of applications in the end-station. A_{ij} specifies the application j of the end-station i. Besides, the criticality level of each application is denoted by $\mathfrak{C}_{ij}$. Each application includes one or more tasks as shown in Eq. (5).

$$A_{ij} := \langle \{\tau_{ij1}, ..., \tau_{ij|A_{ij}|}\}, \mathfrak{C}_{ij} \rangle \tag{5}$$

A task, denoted by τ_{ijk}, has the attributes as shown in Eq. (6):

$$\tau_{ijk} := \langle C_{ijk}, T_{ijk}, O_{ijk}, P_{ijk}, J_{ijk}, B_{ijk}, R_{ijk}, D_{ijk} \rangle \tag{6}$$

where k is the ID of the task. The attributes of a task are specified as follows. C_{ijk} presents the worst-case execution time of the task, T_{ijk} denotes the period, and O_{ijk} represents the offset. Moreover, P_{ijk} is the priority, J_{ijk} is the release jitter, and B_{ijk} is the blocking time experienced by the task. Finally, R_{ijk} denotes the worst-case response time and D_{ijk} presents the deadline of the task. Note that we consider a one-to-one mapping between a SWC (a design-time entity) in the component model and a task (run-time entity) in an end-station in the end-to-end timing model.

B. TSN Network Timing Model

The properties of the network $\mathcal{N}_i$ are as follows.

$$\mathcal{N}_i := \langle s_i, \mathcal{L}_i, \mathcal{I}_i, \mathcal{M}_i, Slope, Preemption \rangle \tag{7}$$

where, s_i is the network speed. $\mathcal{L}_i$ is the set of links connecting the end-stations to TSN switches and the TSN switches to each other. $\mathcal{I}_i$ indicates TSN classes through which the messages are communicated. The set $\mathcal{M}_i$ holds the TSN network's messages. The available TSN classes are shown in Eq. (8).

$$\mathcal{I}_i = \{ST, AVB, BE\} \tag{8}$$

where, ST is the highest priority traffic (Scheduled Traffic) which is scheduled offline with offsets. The class AVB (Audio-Video Bridging) associates multiple priorities (up to maximum 8 priorities) to the TSN port's queues which undergo the CBS mechanism. In AVB class, priorities are specified in

alphabetical order and can be assigned to all the egress queues. For example, class A has a higher priority than class B. Class BE is the lowest priority class (Best-effort) which often does not have any timing requirements. Each queue in TSN port can be set to operate under only one of these classes.

A message belonging to the set of messages $\mathcal{M}_i$ is denoted by m_{ij}, where i shows the ID of the network the message belongs to and j is the ID of the message within that network. The definition of the message in the end-to-end timing model maps to the same entity in the component model.

$$\mathrm{m}_{ij} := \langle \mathcal{X}_{ij}, C_{ij}, P_{ij}, Size_{ij}, T_{ij}, \mathcal{L}_{ij}, J_{ij}, B_{ij}, R_{ij}, \mathcal{O}_{ij}, D_{ij} \rangle \quad (9)$$

where, $\mathcal{X}_{ij}$ is the transmission mode which can be periodic or sporadic. C_{ij} is the worst-case transmission time, P_{ij} is the priority, and $Size_{ij}$ is the payload size. The period is shown by T_{ij}. In the case of sporadic transmission, T_{ij} represents the minimum inter-arrival time between any two instances of the message. The set of links in the route of the message is shown by $\mathcal{L}_{ij}$. The release jitter is shown by J_{ij} and B_{ij} denotes the blocking time which contributes to calculating the response time shown by R_{ij}. Moreover, $\mathcal{O}_{ij}$ is the offset of the message in case the priority of the message is ST, and finally, D_{ij} represents the deadline of the message.

The set of slopes denoted by $Slope$ holds the set of idle slopes for each AVB port queue connected to a link in the network ($\mathcal{L}_i$). $Slope$ values are used to allocate bandwidth for each queue assigned to AVB class. Moreover, the $Preemption$ set holds the set of flags per link in the network ($\mathcal{L}_i$). This list specifies whether the corresponding link to each member of the set allows pausing transmission of an ongoing frame in favor of a higher priority frame.

C. Transactional Model

For the sake of end-to-end analysis, the timing model follows the transactional model in [36] to link the instances of the tasks and messages into one transaction. A set of transactions is indicated by Γ, and a transaction, denoted by Γ^d, contains a chain of tasks and messages that allow flow of data within the system. The superscript d indicates the transaction ID. The chain can either contain several tasks that are within one end-station or a set of tasks distributed over multiple end-stations that communicate through the network. An end-station may include one or more of its tasks in the transaction. The attributes of a transaction are shown in Eq. (10).

$$\Gamma^d := \left\langle Chain^d, T^d, a^d, r^d, Cr^d \right\rangle \quad (10)$$

where the period of the transaction is T^d. The notations r^d and a^d are the transaction's reaction time and data age delays that are calculated by the end-to-end data-propagation delay analysis. Besides, Cr^d specifies the set of timing constraints on the reaction time and data age delays of the transaction. $Chain^d$ represents the chain of tasks in transaction Γ^d as shown in Eq. (11):

$$Chain^d := (\{\tau^d_{ij\alpha}, \ldots, \tau^d_{ij\Omega}\}, m^d_{i1}, \{\tau^d_{nj\alpha}, \ldots, \tau^d_{nj\Omega}\}, \ldots, m^d_{ix}, \{\tau^d_{kj\alpha}, \ldots, \tau^d_{kj\Omega}\}) \quad (11)$$

In Eq. (11), the order of tasks from left to right shows the direction of the data flow, which starts from the initiator task ($\tau^d_{ij\alpha}$), and ends in the terminator task ($\tau^d_{kj\Omega}$) of the transaction.

A transaction can comprise of a set of tasks of one end-station. In such a case, this end-station is assumed to be initiator and terminator of the transaction. Moreover, a transaction may comprise of tasks that are located on different end-stations, then two adjoining tasks in the transaction belonging to two different end-stations communicate via network messages. All the tasks in the transaction that belong to the transaction's initiator end-station ($\mathcal{E}_i$) are specified at the beginning of the transaction. Similarly, all the tasks belonging to the transaction's terminator end-station ($\mathcal{E}_k$) are specified at the end of the transaction. There can be one or more intermediary end-stations, such as ($\mathcal{E}_n$) in Eq. (11), that are part of the transaction. The first message in transaction is sent from the initiator end-station. The message sent from each end-station is placed at the right hand side of the set of end-station's tasks in the transaction, immediately after the source task.

In a TSN network, a message can pass through a set of links, which are specified by the set ($\mathcal{L}_{ij}$). Since, multiple end-stations can be engaged in a transaction, multiple messages must be communicated in a transaction. We assume that there can be a set of x number of messages in the transaction. The terminator end-station receives the last message, by the left most task in its set of tasks. Since, tasks and messages can be used by multiple transactions the superscript d shows the transaction ID that uses the task or message. In Eq. (11), m^d_{ix} denotes the message x that is activated by end-station i, and is involved in the data flow represented by chain d.

D. Timing Requirements Model

The timing constraint on the transaction, Cr^d, defines the maximum allowed value of the delays, such as data age (Age^d), and reaction time ($Reac^d$). These constraints are shown in Eq. (12) for the transaction Γ^d:

$$Cr^d := \{Age^d, Reac^d\} \quad (12)$$

IV. Proposed End-to-end Timing Model Extraction Methodology

The proposed methodology aims at automated extraction of end-to-end timing information from component-based software architectures of TSN-based distributed embedded systems. The extracted timing information is populated in an instance of the end-to-end timing model, which is fed as input to the end-to-end data-propagation analysis engines. An end-user, also known as the software architecture developer/modeler, is able to interact with the component model directly by manually modeling the software architecture. The software architecture can be developed using any component model. As a proof of concept, we use RCM to model the software architecture. The end-user often has limited or no knowledge about detailed timing information in the system, but is skilled in designing software architectures using component models. In order to retrieve the required timing information to analyze

and verify the timing requirements on the software architectures, we propose a timing model extraction methodology that is conceptualized in Fig. 2.

The *end-user* shown in module *a* in Fig. 2 develops the software architecture with the help of the component models as shown in module *b* in Fig. 2. Accordingly, the *end-to-end timing model extraction* shown in module *c* in Fig. 2, coordinates the extraction of an instance of the end-to-end timing model from the software architecture. Some information can be implicitly obtained from the software architecture, hence there is no need for detailed specification of such information by the software architecture developer. Furthermore, some information includes variable parameters that can be refined in the software architecture through the *configuration* step, as shown by module *b2* in Fig. 2. The configuration step is performed while developing the software architecture. Besides, this step could also be iterated by the system's integrator/configurator or by the automated configuration tools.

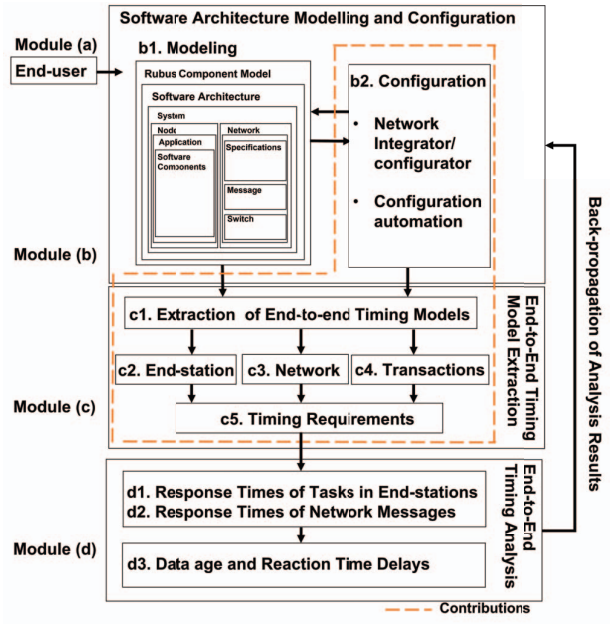

Fig. 2: Timing model extraction methodology.

The module *c1* extracts the end-to-end timing information from the model of distributed software architecture that is augmented with configuration information. The sources of end-to-end timing information can be classified as follows:

1) **User-defined (User):** The properties in this category are extracted from the input specified by the end-user while developing the software architecture. The end-user may not have knowledge about the detailed timing aspects of the system.
2) **Software-architecture-derived (SWA-d):** This timing information includes the properties that are either inherited from other components in the software architecture, calculated according to the user-defined properties or implicitly initialized based on other properties set by other sources.
3) **Configurable (Conf.):** This category holds the properties obtained from the software architecture, which are configured according to some logical constraints, and algorithms for the sake of optimizing timing performance of the system. Such parameters could also be defined by system experts based on their knowledge of the system requirements, i.e., system configurators or integrators.
4) **Analysis-derived (Analysis):** The values of the properties in this category are obtained by performing various analyses, e.g., response-time analysis of individual end-stations, response-time analysis of TSN network, and end-to-end data-propagation analysis.

In the end-to-end timing model extraction step, module *c1* in Fig. 2, the properties obtained from various sources in module *b*, i.e., the software architecture developer, software architecture and configurators, are mapped to the parameters in the proposed end-to-end timing model as shown in Table I. In the next subsections, we explain the end-to-end timing model specification for each entity, as shown in modules *c2*, *c3*, *c4* and *c5* of Fig. 2.

Finally, the extracted end-to-end timing model is fed into the *end-to-end timing analysis* as shown in module *d* in Fig. 2. Furthermore, under the procedures in module *d1* and *d2* in Fig. 2 the response times of the tasks and messages, and subsequently the end-to-end delays, i.e., data age and reaction time delays are calculated under module *d3* in Fig. 2. Subsequently, the results of the end-to-end timing analysis are propagated back to the distributed software architecture model.

A. Specification of End-station Timing Information

In this step, each extracted property from the node component is specified to the associating entity, namely end-station in the proposed end-to-end timing model.

The software architecture developer (end-user) designs the software architecture's hierarchy, which includes the connections between the end-stations and the arrangement of the SWCs within an end-station. Therefore, an application (A_{ij}) and assignment of the tasks to each application is done by the software architecture developer. The criticality level ($\mathfrak{C}_{ij}$) of each application is associated to the priority of the tasks included in the application therefore the criticality level of the application is derived from the software architecture according to the priority of the tasks included in this application.

The parameters of the end-station that can be set directly by the end-user include the worst-case execution time (C_{ijk}) and the task priority (P_{ijk}). These user-defined parameters can be further used for obtaining the rest of the parameters in the classes software-architecture-derived, configurable and analysis-derived.

In case of considering implicit deadlines in the system, the deadline of the task (D_{ijk}) is equal to the task's period. As a result, deadline value is derived from the software architecture. The deadline can also be assigned according to the system timing information after performing the end-to-end timing analysis engines. Therefore, the deadline parameter can be also considered as configurable parameter.

Task's period (T_{ijk}) assignment is dependent on whether the task is independently or dependently triggered. If the task is

TABLE I: Extracting the proposed timing model for TSN.

Component	Proposed Timing Model (*c*, Fig. 2)	Proposed Parameters	User	SWA-d	Conf.	Analysis
Node	End-station (*c2*, Fig. 2)	A_{ij}	✓			
		$\mathfrak{C}_{ij}$		✓		
SWC	Task (*c2*, Fig. 2)	C_{ijk}	✓			
		T_{ijk}	✓	✓		
		O_{ijk}		✓	✓	
		P_{ijk}	✓			
		J_{ijk}				✓
		B_{ijk}				✓
		R_{ijk}				✓
		D_{ijk}		✓	✓	
Network	Network (*c3*, Fig. 2)	s_i		✓		
		$\mathcal{L}_i$		✓		
		$\mathcal{I}_i$		✓		
		$Slope$		✓	✓	
		Preemption		✓	✓	
Message	Message (*c3*, Fig. 2)	$\mathcal{X}_{ij}$		✓		
		C_{ij}		✓		
		P_{ij}		✓		
		$Size_{ij}$	✓			
		T_{ij}		✓		
		$\mathcal{L}_{ij}$	✓	✓	✓	
		J_{ij}				✓
		B_{ij}				✓
		R_{ij}				✓
		$\mathcal{O}_{ij}$		✓	✓	
		D_{ijk}		✓	✓	
SWC Chain	Transaction (*c4*, Fig. 2)	Γ^d	✓			
		$Chain^d$	✓			
		T^d		✓		
Requirement	Requirement (*c5*, Fig. 2)	Cr^d	✓		✓	
		a^d				✓
		r^d				✓

independently triggered, then the task's period is user-defined. If the task is dependently triggered, the period of the task is inherited from its predecessor entity (task or message), which triggers the task.

Task's offset (O_{ijk}) is defined according to the pseudo code in Algorithm 1. The offset of a task is a configurable property in case the task is sending/receiving a message through the ST class in TSN. If the task is not connected to the network, task offset is derived from the software architecture. Task offset influences the quality of service of the system. The offsets can be defined in a manner to reduce the end-to-end delays (age and reaction) of the transactions. Besides, there are various other algorithms in literature which consider different optimization criteria for setting offsets of the tasks in distributed systems [37].

Finally, the release jitter (J_{ijk}), blocking (B_{ijk}) and response time (R_{ijk}) of the task are retrieved from timing analysis techniques and corresponding engines, such as [38].

Algorithm 1: Specifying task offset.

```
for each Γ^d ∈ Γ do
    for each τ^d_ijk ∈ Γ^d do
        if τ^d_ijk ∈ initiator E_i then
            if τ^d_ijk.O_ijk is configurable then
                τ^d_ijk.O_ijk is manually set or automatically
                  optimized.
                ST class is used.
            else if τ^d_ijk.O_ijk is SWA-d then
                No offset is needed for the task.
                Release the task at time 0.
                A non-ST class is used.
        else if τ^d_ijk ∈ terminator E_i then
            if τ^d_ijk receives ST messages then
                τ^d_ijk.O_ijk manually set or automatically
                  optimized according to the received
                  message's offset at its last link.
            else if τ^d_ijk receives non-ST messages then
                No offset is needed for the task. Release the
                  task at time 0.
```

B. Specification of Network Timing Information

In case of the network, the network speed (s_i), set of links ($\mathcal{L}_i$) and the available TSN classes ($\mathcal{I}_i$) are properties that are dependent on the design of the network within the software architecture.

The set of idle slopes (e.g. $slope_A$ and $slope_B$) can be chosen simply by globally assigning a value that applies to all the links in the TSN network, or it can be individually assigned for all the links that use AVB traffic, e.g., according to the load of AVB queues on each of the links. Consequently, the configuration of the idle slopes requires to deal with a trade-off between the simplicity of configuring the network versus the performance achieved by the configuration of the network.

Finally, the $Preemption$ is the last configurable parameter set which can be enabled/disabled to optimize the utilization of the links by the lower priority messages [39]. The pseudo code in Algorithm 2 shows the strategy to define the idle slopes and the $Preemption$ set on the TSN links.

We assume that TSN messages inherit properties such as transmission mode ($\mathcal{X}_{ij}$), period (T_{ij}) (for Event triggered chains) and priority (P_{ij}) from the message's sender task in the transmitter end-station. The deadline of the message (D_{ij}) is assigned implicitly equal to its period. The deadline of the messages can be configured according to the same assumption for the tasks' deadline. The payload size ($Size_{ij}$) of the message is derived from the software architecture since it is calculated based on the payload size and the network speed. The worst-case transmission time (C_{ij}) of the message is derived from the software architecture based on payload size ($Size_{ij}$) of the message, and network speed (s_i).

The set of links that the message passes from the transmitter end-station to reach the receiver end-station, denoted by $\mathcal{L}_{ij}$, are derived from the software architecture. In some cases, there

is only one route that delivers the message to the destination end-station, therefore it is possible for the software architecture developer to specify the routes manually. However, in many cases, there are more than one route from the transmitter end-station to the destination end-station. Consequently, the set of links that the message traverses can be configured to optimize the routing of the message. For example, some works define routes simply based on the shortest path [40]. On the other hand, TSN messages can also be re-routed with respect to the potential failure of the links [41] or according to the load of each traffic class in the network, i.e, ST or AVB [42]. Hence, the set of routes ($\mathcal{L}_{ij}$) is also specified as a configurable parameter.

As the pseudo code in Algorithm 3 indicates, the message offsets per link, denoted by the set $\mathcal{O}_{ij}$, are defined according to the priority of the message. If the message is assigned to class ST, there is a set of offsets per links between the source to the destination of the message. This set can be configured based on different optimization approaches. For instance, the optimization algorithm presented in [40] schedules the ST traffic in a way to reduce the end-to-end delay in the non-ST traffic. The optimization approaches for scheduling the ST traffic can be found in a recent survey [43]. If the message is not ST, the set of offsets per link ($\mathcal{O}_{ij}$) will not be used and contains all zeroes.

Algorithm 2: Specifying slope and preemption.

```
if simple link setup is desired then
    Set the idle slopes and preemption GLOBALLY.
else if optimized link setup is desired then
    Set the idle slopes and preemption PER LINK.
```

C. Specification of Transaction Timing Information

The transactions are specified by the software architecture developer in order to analyze a chain of tasks and messages ($Chain^d$) in the system. Tasks in a transaction can be triggered in two modes, i.e., "independent" or "dependent". Independently triggered tasks are activated based on their individual clocks or trigger sources, whereas dependently triggered tasks are activated by their predecessor tasks. The trigger mode for the messages are different depending on the TSN class defined for the message. If the message is ST, it is triggered independently based on static offsets defined for it per link specified in its route to the destination end-station. In case of the dependent trigger mode, the message is triggered right after the execution of its predecessor task is finished. Messages assigned to non-ST classes, such as AVB or BE, are activated dependently.

The transaction period (T^d) is calculated by finding the Least Common Multiple (LCM) of the periods of all tasks within the transaction.

D. Specification of Timing Requirements

All the timing constraints in the transaction, denoted by Cr^d, and end-to-end deadline can be either specified by the software architecture developer when defining the transaction; or they can be configured according to the network/system analysis. The reaction time (r^d) and data age (a^d) delays of the transaction d are calculated by the end-to-end data-propagation delay analysis.

Algorithm 3: Specifying message offsets.

```
if m_ij.P_ij is ST then
    Message offset (m_ij.O_ij) is configurable.
    Optimize message offsets per link.
else if m_ij.P_ij is not ST then
    SWA-d mode.
    Message has no offset.
```

V. Evaluation on a Vehicular Industrial Use-case

The end-to-end timing model extraction methodology presented in the previous section is implemented as a proof of concept in the Rubus-ICE tool suite. In this section, we take advantage of a real vehicular use-case to validate the applicability of the proposed end-to-end timing model extraction methodology. For the sake of evaluations, we first model a software architecture of a distributed embedded system, consisting of a set of transactions, in RCM. Then we extract the end-to-end timing model from the software architecture in Rubus-ICE. Using the extracted timing model, we perform the end-to-end data-propagation delay analysis of the software architecture using the implemented analysis in Rubus-ICE [35] and discuss the analysis results.

A. Use-case Setup

The network topology in the use-case consists of nine end-station that are interconnected via three TSN switches as depicted in Fig. 3. In the topology, all the end-stations are capable of generating TSN traffic, whereas HU and AVSink only receive TSN traffic from the other end-stations. The set of transactions and network traffic in this use-case are presented in Table II. In summary, there are a set of 10 distributed chains (transactions). Each transaction includes tasks from two different end-stations and one message between the end-stations. The initiator end-station has only one task that sends the message to the network. The transaction terminator end-station includes maximum two tasks. On the receiver's side, the first task receives the TSN message and activates the second task in the receiver end-station. We assume that all tasks in an end-station belong to the same application.

The transactions 1 to 4 are using ST class and the transactions 5 to 10 are using BE class in the network. The offset of the sender tasks in transactions 1 to 4 are set to the default value which is 0. Accordingly, the messages transmitted from these tasks are assigned to ST class. Finally, the idle slopes are set to 0 for all the links since the AVB classes are not used in the use-case. The reaction time (Reac_d) and data age (Age_d) constraints on all the transactions are subsequently 70 ms and 45 ms. These constraints are specified by expert integrators from the industry (providers of the use-case).

TABLE II: Evaluation settings for the use-case based on the distributed chains (transactions). All times are in milliseconds.

Γ^d	Source ($\mathcal{E}_i$)	Source tasks (τ^d_{ijk}): [id, P_{ijk},C_{ijk}, T_{ijk}]	Message (m^d_{jk}): [id, P_{jk},$Size_{jk}$, T_{jk}, $\mathcal{O}^r_{jk}$]	Destination ($\mathcal{E}_i$)	Destination tasks (τ^d_{ijk}): [id, P_{ijk},C_{ijk}, T_{ijk}]	
		Task 1			Task 2	Task 3
1	Cloud Gateway (3)	[$\tau^1_{3,1,1}$,1,0.05,10]	[$m^1_{1,1}$, ST,1542,10,0.038]	AVSink (8)	[$\tau^1_{8,1,1}$,5,0.05,10]	[$\tau^1_{8,1,2}$,4,0.05,10]
2	Remote Control (2)	[$\tau^2_{2,1,1}$,3,0.05,10]	[$m^2_{1,2}$, ST,1542,10,0.038]	HU (7)	[$\tau^2_{7,1,1}$,5,0.05,10]	[$\tau^2_{7,1,2}$,4,0.05,10]
3	Camera (9)	[$\tau^3_{9,1,1}$,2,0.05,10]	[$m^3_{1,3}$, ST,1542,10,0.013]	HU (7)	[$\tau^3_{7,1,1}$,5,0.05,10]	[$\tau^3_{7,1,2}$,4,0.05,10]
4	GMSL Camera (6)	[$\tau^4_{6,1,1}$,1,0.05,10]	[$m^4_{1,4}$, ST,1542,10,0.025]	AVSink (8)	[$\tau^4_{8,1,1}$,5,0.05,10]	[$\tau^4_{8,1,2}$,4,0.05,10]
5	Camera (9)	[$\tau^5_{9,1,2}$,1,0.05,20]	[$m^5_{1,5}$, BE,1542,20,0]	AVSink (8)	[$\tau^5_{8,1,3}$,3,0.05,20]	[$\tau^5_{8,1,4}$,2,0.05,20]
6	Remote Control (2)	[$\tau^6_{2,1,2}$,2,0.05,10]	[$m^6_{1,6}$, BE,1542,10,0]	AVSink (8)	[$\tau^6_{8,1,3}$,3,0.05,20]	[$\tau^6_{8,1,4}$,2,0.05,20]
7	I/O (1)	[$\tau^7_{1,1,1}$,1,0.05,10]	[$m^7_{1,7}$, BE,1542,10,0]	AVSink (8)	[$\tau^7_{8,1,5}$,1,0.05,10]	
8	Machine Control (4)	[$\tau^8_{4,1,1}$,1,0.05,10]	[$m^8_{1,8}$, BE,1542,10,0]	AVSink (8)	[$\tau^8_{8,1,3}$,3,0.05,20]	[$\tau^8_{8,1,4}$,2,0.05,20]
9	Remote Control (2)	[$\tau^9_{2,1,3}$,1,0.05,10]	[$m^9_{1,9}$, BE,1542,10,0]	HU (7)	[$\tau^9_{7,1,3}$,3,0.05,10]	[$\tau^9_{7,1,4}$,2,0.05,10]
10	High-level Control (5)	[$\tau^{10}_{5,1,1}$,1,0.05,10]	[$m^{10}_{1,10}$, BE,1542,10,0]	HU (7)	[$\tau^{10}_{7,1,3}$,3,0.05,10]	[$\tau^{10}_{7,1,5}$,1,0.05,10]

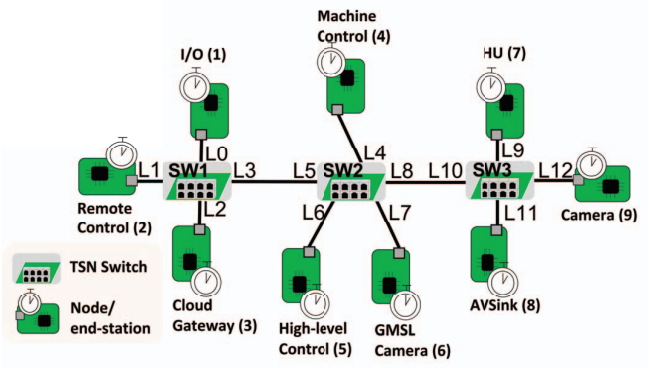

Fig. 3: Use-case from the vehicular domain.

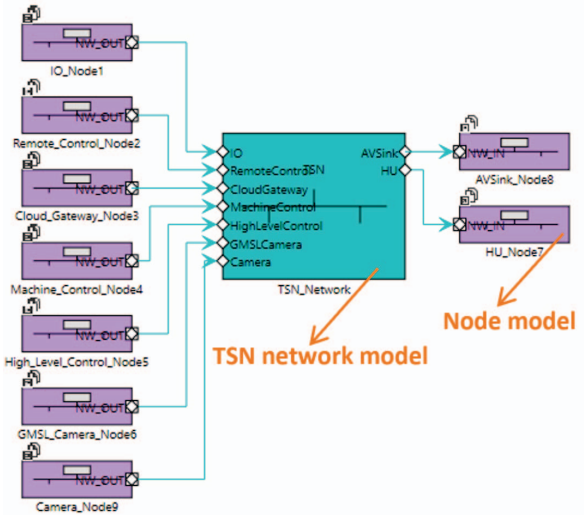

Fig. 4: System-level software architecture of the use-case.

B. Modeled Use-case in Rubus-ICE

The system-level software architecture of the use-case modeled with RCM is depicted in Fig. 4. The system-level software architecture consists of nine node models that correspond to the nine end-stations. All the nodes are connected to one TSN network model, as shown in Fig. 4. In the internal model of the TSN network as shown in Fig. 5, the flow from all the sender nodes are either towards HU or AVSink (the only sink nodes within the system).

The internal software architecture of each node is depicted in Fig. 6, where the RCM representation of the set of SWCs within each node is shown by yellow components. For example, the software architecture of the Camera node consists of two SWC, where SWC1 in the Camera node that is used in transaction 3 (represented by $\tau^3_{9,1,1}$ in the end-to-end timing model) has period of 10 ms and sends an ST message with the priority of 2. Besides, SWC2 in the Camera node is used in transaction 5 (represented by $\tau^5_{9,1,2}$ in the end-to-end timing model) and has a lower priority than SWC1 (priority 1). The period of SWC2 is 20 ms. Transaction 3 is initiated from the SWC1 ($\tau^3_{9,1,1}$) in the Camera node and is terminated at the SWC2 in the HU node ($\tau^3_{7,1,2}$), which has the priority of 4. The terminator task of the transaction ($\tau^3_{7,1,2}$) is triggered by the SWC1 in the HU node ($\tau^3_{7,1,1}$). As a result, the terminator task inherits the period of its predecessor task, namely SWC1 in the HU node ($\tau^3_{7,1,1}$).

C. End-to-end Response-Time Analysis Results

Table III presents the results of the end-to-end data-propagation delay analysis. The results include reaction time and data age delays of the transactions as well as response times of the network messages. For instance, transaction 3 contains an ST message with the response time of 0.025 ms. The data age and reaction time delays of transaction 3 are subsequently 20.100 and 30.100 ms. Transaction 5 is also initiated at the Camera node, though from a different SWC, namely (SWC2), and it is terminated at SWC4 in HU. Transaction 5 uses the class BE of the TSN network. The calculated response time of the message in Transaction 5 is 0.153 ms. Besides, the data age and reaction time delays of transaction 5 are subsequently 40.300 and 60.300 ms. As the specified data age and reaction time constraints on all transactions are 70 ms and 45 ms respectively, it is evident from Table III that all transactions meet their specified timing constraints.

VI. Conclusions

In this paper, we presented a detailed end-to-end timing model of vehicular distributed embedded systems that utilize Time-Sensitive Networking (TSN) standards. This timing

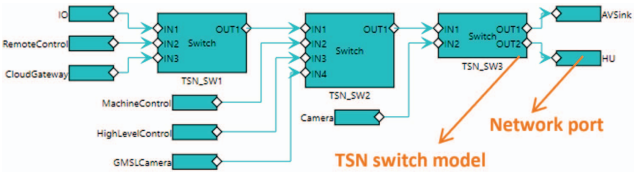

Fig. 5: The TSN network model of the use-case.

TABLE III: Calculated data age and reaction time delays, and message response times for each transaction.

Trans. (Γ^d)	$\Gamma^d.r^d$ (ms)	$\Gamma^d.a^d$ (ms)	$m_{ij}.R_{ij}$ (ms)
1	30.100	20.100	0.050
2	30.100	20.100	0.050
3	30.100	20.100	0.025
4	30.100	20.100	0.038
5	60.300	40.300	0.153
6	50.300	30.300	0.332
7	20.550	10.550	0.332
8	50.300	30.300	0.178
9	30.300	20.300	0.332
10	30.300	20.300	0.255

model is required as a necessary input for the timing analysis engines. We also proposed a methodology for automated extraction of timing information for the instances of this model from the software architectures of the systems. Hence, the proposed methodology facilitates automated end-to-end data-propagation delay analysis of the software architectures of these systems. As a proof-of-concept, we implemented the end-to-end timing model and the model extraction methodology in an industrial tool suite, namely Rubus-ICE. We evaluated the proposed methodology using a vehicular industrial use-case. In this regard, we modelled the software architecture of the use case with an industrial component model (RCM) and performed its timing analysis using the proposed methodology in Rubus-ICE tool suite. The results demonstrate the applicability of the proposed model and methodology. The proposed methodology can be adapted for other component models that use the principles of model- and component-based software development and use the pipe-and-filter communication among software components such as AUTOSAR. One future research direction is to consider back-propagation of the analysis results for refining the software architecture and re-configuring the models of TSN networks.

Acknowledgment: This work is supported by the Swedish Governmental Agency for Innovation Systems (VINNOVA) via the DESTINE, INTERCONNECT and PROVIDENT projects and by the Swedish Knowledge Foundation via the projects DPAC & HERO. We would like to thank all industrial partners, in particular HIAB, Arcticus Systems and Volvo CE.

REFERENCES

[1] L. Lo Bello, R. Mariani, S. Mubeen, S. Saponara, "Recent Advances and Trends in On-Board Embedded and Networked Automotive Systems," *IEEE Transactions on Industrial Informatics*, November 2019.

[2] *802.1Q-2022 - IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks*. IEEE, 2022.

[3] IEEE, *IEEE Std. 802.1Qbv, IEEE Standard for Local and Metropolitan Area Network–Bridges and Bridged Networks, Amendment 25: Enhancement for Scheduled Traffic*, 2015.

[4] *IEEE Std. 802.1Qbu-2016: IEEE Standard for Local and Metropolitan Area Network–Bridges and Bridged Networks, Bridges and Bridged Networks - Amendment 26: Frame Preemption*, IEEE, 2016.

[5] M. Ashjaei, L. Lo Bello, M. Daneshtalab, G. Patti, S. Saponara, S. Mubeen, "Time-Sensitive Networking in Automotive Embedded Systems: State-of-the-Art and Research Opportunities," *Journal of Systems Architecture*, August 2021.

[6] T. Vale, I. Crnkovic, E. S. de Almeida, P. A. da Mota Silveira Neto, Y. Cerqueira Cavalcanti, S. R. de Lemos Meira, "Twenty-Eight Years of Component-Based Software Engineering," *Journal of Systems & Software*, 2016.

[7] T. A. Henzinger, J. Sifakis, "The Embedded Systems Design Challenge," in *14th International Symposium on Formal Methods*, 2006.

[8] AUTOSAR Consortium, AUTOSAR Techincal Overview, Release 4.1, Rev.2, Ver.1.1.0., http://autosar.org.

[9] K. Hanninen, J. Maki-Turja, M. Nolin, M. Lindberg, J. Lundback, K. Lundback, "The Rubus Component Model for Resource Constrained Real-Time Systems," in *IEEE Symposium on Industrial Embedded Systems*, 2008.

[10] C. Wolff , L. Krawczyk, R. Höttger, C. Brink, U. Lauschner, D. Fruhner, E. Kamsties, B. Igel, "AMALTHEA — Tailoring Tools to Projects in Automotive Software Development," in *2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, 2015.

[11] "EAST-ADL Domain Model Specification, V2.1.12,," http://www.east-adl.info/Specification/V2.1.12/EAST-ADL-Specification_V2.1.12.pdf.

[12] P.H. Feiler, D.P. Gluch, J.J. Hudak, "The Architecture Analysis & Design Language (AADL): An introduction," Tech. Rep., 2006.

[13] N. Feiertag, K. Richter, J. Nordlander, 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*, 2008.

[14] M. Becker, D. Dasari, S. Mubeen, M. Behnam, T. Nolte, "End-to-end Timing Analysis of Cause-Effect Chains in Automotive Embedded Systems," *Journal of Systems Architecture*, 2017.

[15] S. Mubeen, T. Nolte, M. Sjödin, J. Lundbäck, K. Lundbäck, "Supporting Timing Analysis of Vehicular Embedded Systems through the Refinement of Timing Constraints," *International Journal on Software and Systems Modeling*, 2017.

[16] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, R. Ernst, "System-level Performance Analysis - the SymTA/S Approach," *Computers and Digital Techniques*, 2005.

[17] S. Mubeen, J. Mäki-Turja, 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*, January 2013.

[18] S. Mubeen, H. Lawson, J. Lundbäck, M. Gålnander, K. Lundbäck, "Provisioning of Predictable Embedded Software in the Vehicle Industry: The Rubus Approach," in *4th International Workshop on Software Engineering Research and Industry Practice*, 2017.

[19] S. Mubeen, M. Ashjaei, M. Sjödin, "Holistic Modeling of Time Sensitive Networking in Component-based Vehicular Embedded Systems," in *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2019.

[20] A. Arestova, M. Martin, K-S J. Hielscher, R. German, "A Service-oriented Real-time Communication Scheme for AUTOSAR Adaptive using OPC UA and Time-sensitive Networking," *Sensors*, 2021.

[21] A. Bucaioni, L. Addazi, A. Cicchetti, F. Ciccozzi, R. Eramo, S. Mubeen, M. Sjödin, "MoVES: A Model-Driven Methodology for Vehicular Embedded Systems," *IEEE Access*, 2018.

[22] A. Bucaioni, M. Becker, J. Lundbäck, H. Mackamul, "From AMALTHEA to RCM and Back: a Practical Architectural Mapping Scheme," in *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2020.

[23] A. Bucaioni, V. Dimic, H. Lönn, M. Gålnander, J. Lundbäck, "Transferring a Model-based Development Methodology to the Automotive Industry," in *22nd IEEE International Conference on Industrial Technology*, 2021.

[24] S. Mubeen, M. Gålnander, J. Lundbäck, K. Lundbäck, "Extracting Timing Models from Component-based Multi-criticality Vehicular Embedded Systems," in *15th International Conference on Information Technology : New Generations*, 2018.

[25] ISO 11898-1, "Road Vehicles–Interchange of Digital Information–Controller Area Network (CAN) for High-speed Communication, ISO Standard-11898, November 1993."

[26] "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.

[27] "AUTOSAR Requirements on Communication, Release 4.2.1," www.autosar.org, accessed on March 15, 2019.

[28] B. Houtan, M-O. Aybek, M. Ashjaei, M. Daneshtalab, M. Sjödin, S. Mubeen, "End-to-end Timing Model Extraction from TSN-Aware

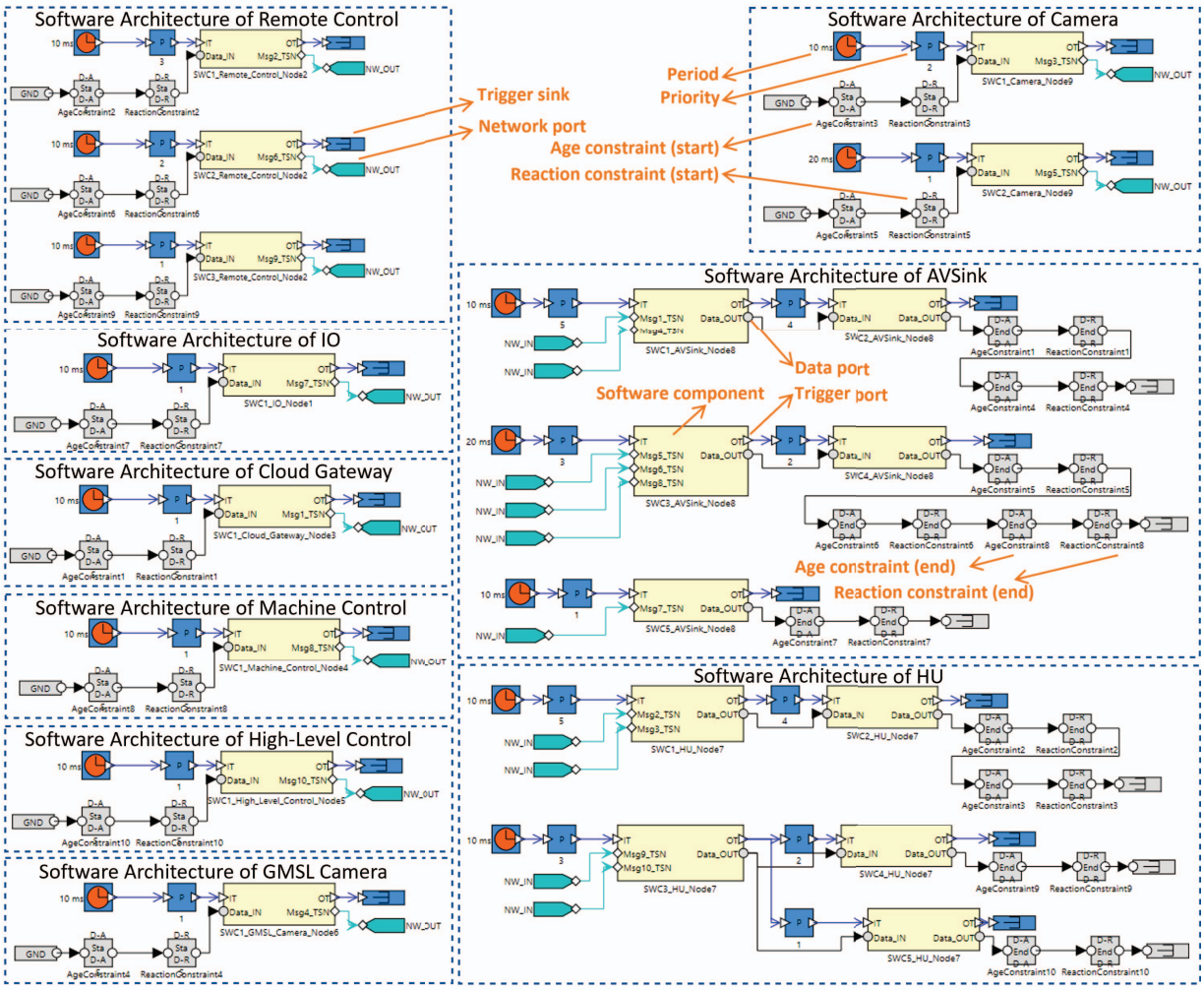

Fig. 6: Software architecture of the nodes along with the timing constraints specified on ten distributed chains in RCM.

Distributed Vehicle Software," in *2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2022.

[29] F. Stappert, J. Å. Jönsson, J. Mottok, R. Johansson, "A Design Framework for End-To-End Timing Constrained Automotive Applications," in *Embedded Real Time Software and Systems Conference*, 2010.

[30] M. Becker, Matthias, D. Dasari, S. Mubeen, M. Behnam, T. Nolte, "Synthesizing Job-level Dependencies for Automotive Multi-rate Effect Chains," in *2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications*, 2016.

[31] M. Ashjaei, N. Khalilzad, S. Mubeen, M. Behnam, I. Sander, L. Almeida, T. Nolte, "Designing End-to-end Resource Reservations in Predictable Distributed Embedded Systems," *Real-Time Sys.*, 2017.

[32] S. Mubeen, J. Mäki-Turja, M. Sjödin, "Communications-Oriented Development of Component-Based Vehicular Distributed Real-Time Embedded Systems," *Journal of Systems Architecture*, 2014.

[33] M. Ashjaei, S. Mubeen, J. Lundbäck, M. Gålnander, K-L. Lundbäck, T. Nolte, "Modeling and Timing Analysis of Vehicle Functions Distributed over Switched Ethernet," in *43rd Annual Conference of the IEEE Industrial Electronics Society*, 2017.

[34] S. Mubeen, M. Gålnander, A. Bucaioni, J. Lundbäck, K-L. Lundbäck, "Timing Verification of Component-based Vehicle Software with Rubus-ICE: End-user's Experience," in *IEEE/ACM 1st International Workshop on Software Qualities and their Dependencies*, 2018.

[35] B. Houtan, M. Ashjaei, M. Daneshtalab, M. Sjödin, S. Mubeen, "Supporting End-to-end Data-propagation Delay Analysis for TSN Networks," Tech. Rep., 2021.

[36] J.C. Palencia, M. Gonzalez Harbour, "Schedulability Analysis for Tasks with Static and Dynamic Offsets," *Proceedings 19th IEEE Real-Time Systems Symposium*, 1998.

[37] A. Minaeva, Z. Hanzálek, "Survey on Periodic Scheduling for Time-Triggered Hard Real-Time Systems," *Association for Computing Machinery*, 2022.

[38] B. Houtan, M. Ashjaei, M. Daneshtalab, M. Sjödin, S. Afshar, S. Mubeen, "Schedulability Analysis of Best-effort Traffic in TSN Networks," in *26th IEEE International Conference on Emerging Technologies and Factory Automation*, 2021.

[39] J. Cao, M. Ashjaei, P. J. Cuijpers, R. J. Bril, J. Lukkien, "Independent Yet Tight WCRT Analysis for Individual Priority Classes in Ethernet AVB," in *International Workshop on Factory Communication Systems*, 2018.

[40] Bahar Houtan, Mohammad Ashjaei, Masoud Daneshtalab, Mikael Sjödin, Saad Mubeen, "Synthesising Schedules to Improve QoS of Best-effort Traffic in TSN Networks," in *29th International Conference on Real-time Networks and Systems*, 2021.

[41] F. Pozo, G. Rodriguez-Navas, H. Hansson, "Schedule Reparability: Enhancing Time-triggered Network Recovery upon Link Failures," in *IEEE International Conference on Embedded and Real-time Computing Systems and Applications*, 2018.

[42] V. Gavriluţ, L. Zhao, M. L. Raagaard, P. Pop, "AVB-aware Routing and Scheduling of Time-triggered Traffic for TSN," *IEEE Access*, 2018.

[43] S. L. M. M. T. Stüber, L. Osswald, "A Survey of Scheduling in Time-Sensitive Networking (TSN)," *arXiv preprint arXiv:2211.10954*, 2022.