An Evaluation Framework for Modeling Languages Supporting Predictable Vehicular Software Systems

Enxhi Ferko, Igli Jasharllari, Alessio Bucaioni, Mohammad Ashjaei, Saad Mubeen
Mälardalen University, Västerås, Sweden
name.surname@mdh.se

***Abstract*—Handling the software complexity of modern vehicular systems has become very challenging due to their non-centralized nature and real-time requirements that they impose. Among many software development paradigms for these systems, model-based development excels for several reasons including its ability to verify timing predictability of software architectures of these systems using pre-runtime timing analysis techniques. In this work, we propose a comprehensive framework that captures the timing related information needed for the modeling languages to facilitate these timing analyses. We validate the applicability of the framework by comparing two modeling languages and their respective tool-chains, Rubus-ICE and APP4MC, that are used for software development in the vehicle industry. Based on our results, both modeling languages support the design and analysis of vehicle software, but with different. Both modeling languages support time-, event- and data-driven activation of software components and modeling of single- and multi-rate transactions. Amalthea targets applications on single nodes with multi-core architectures while RCM focuses on single-core single-node and distributed embedded systems with ongoing work for supporting single-node multi-core architectures. In comparison to Amalthea, RCM provides a generic message model which can easily be re-modeled according to protocol-specific properties.**

***Index Terms*—Model-based software development, model-based design, model-based verification, modelling languages, Rubus Component Model, Amalthea, automotive software, real-time systems, timing analysis, timing verification.**

I. Introduction

The software in modern vehicles consists of millions of lines of code that run on several tens of Electronic Control Units (ECUs) [1], [2]. According to a recent study from Jaguar Land Rover [3], the size of vehicle software will soon reach 1 billion lines of code. To cope with the increasing size and complexity of the software during its development, researchers and practitioners have successfully adopted emerging software development paradigms, including model-based software development [4]. Model-based software development revolves around mete(models) [5], [6] and model transformations [7]. Among numerous benefits of this paradigm is its ability to support model-based timing verification of the software architectures. This is crucial in the vehicle industry as the systems' developers are required to verify the *timing predictability* of these systems. A system is considered timing predictable if it is possible to prove at the design time that all the specified timing requirements are satisfied [8], [9].

In recent years, several modeling languages and methodologies that are supported by the model-based timing analysis have been developed, e.g., the Rubus Component Model (RCM) [10] and Amalthea [11]. However, these languages are characterized by different approaches towards the modeling of timing information (timing properties, requirements and constraints). As a result, the timing information that a timing analysis tool requires as an input may be partially or fully extractable from the software architectures that are modeled with these languages. Consequently, timing analysis of the software architectures that are modeled with the languages that have limited expressiveness for timing information cannot be supported without making over-estimated assumptions.

To date, few researches have focused on characterizing and comparing modeling languages with respect to timing perspective as most of them focus on other extra-functional properties and requirements. In this context, it would be beneficial to both researchers and practitioners to have means for classifying and categorizing modeling languages with respect to modeling timing information and analyses they enable. Such a classification can not only establish a basis for comparing the timing modeling capabilities of the languages, but also identify which of the modeling languages would be more appropriate for a specific vehicular application. We address the following research problem: *How can we categorize and compare modeling languages based on a theoretical framework considering their timing expressiveness?* The main contribution of this work is a comprehensive framework for characterising modeling languages supporting timing analysis of software architectures of vehicular systems. To show the applicability of the proposed framework, we apply it to categorize two industrial modelling languages widely used in the vehicular domain, namely the RCM [10] and Amalthea [11] and their respective tool-chains Rubus-ICE [12] and APP4MC [13].

II. Background and Related Work

A. Rubus Component Model (RCM)

Rubus is a collection methods and tools for model-based software development of real-time embedded systems. RCM, core of the Rubus approach, is a modelling language. It is developed by Arcticus Systems[1] in co-operation with partners from academia and industry. RCM has been used in the vehicular domain for more than 25 years for the development of in-vehicle control functionalities [10]. The Rubus-ICE tool suite consist of modeling and analysis tools, code generators as well as run-time infrastructure for the modeled applications. Within Rubus-ICE, the applications are developed and described graphically as inter-connected components, called Software Circuits (SWCs). The SWCs are characterized by run-to-completion semantics meaning that, upon triggering, an SWC reads the input data, processes it according to the internal

[1] Arcticus Systems - https://www.arcticus-systems.com/

behavioral logic and provides an output. Figure 1 illustrates a multi-rate data chain in RCM which consists of 3 SWCs.

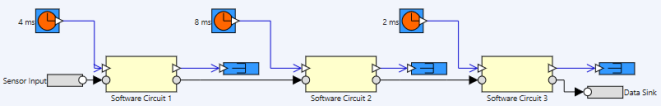

Fig. 1. Multi-rate data chain modeling in RCM.

B. Amalthea model and APP4MC

APP4MC is an open-source European project under Eclipse Public License whose objective is to provide a tool-chain for the engineering of real-time multi-core embedded systems [13]. It originates from the Amalthea and Amalthea4Public ITEA2 research projects [14]. It is still under development as it aims to cover all development cycles of software starting from design to verification and validation of these systems. Amalthea can be seen as an extension to the AUTOSAR [15] standard from various aspects and is envisioned to be a standard for multi-core real-time embedded systems in the vehicular domain [16]. To date, it provides the design and run-time infrastructure of real-time embedded system applications. Systems are designed based on the Amalthea model [11] where the basic hierarchical element is called *Runnable*. A Runnable encapsulates the basic functions. They might be grouped in clusters and mapped to a task which is activated by *Stimuli*. Systems are designed as interconnected components, but yet not graphically supported. Figure 2 illustrates how this is concretely done in Amalthea.

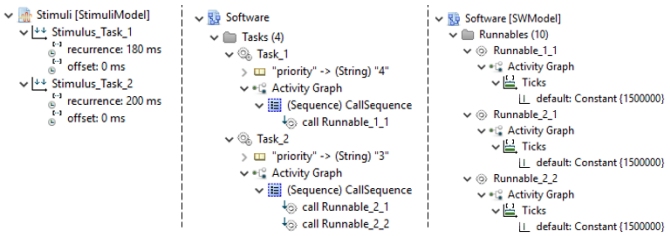

Fig. 2. Runnables and Tasks modeled with Amalthea.

C. Related Work

The research conducted in [4] and [17] presents an overview of the current modeling languages and their respective tool support in the vehicular domain. A comprehensive description regarding the timing modeling and timing analysis is given. The modeling languages and tools are grouped based on the four levels of abstractions for vehicular embedded systems development. In comparison to these works, we aim to develop a comparative evaluation framework for the languages based on their expressiveness to model timing information on the software architectures and timing analyses they enable. A framework for comparing real-time modeling languages based on different aspects such extension capabilities and tool support is presented in [18]. Despite we believe that the properties in [18] are very important when choosing a modeling language, in this work we aim to develop a comprehensive framework with in-depth investigation of their capabilities for describing the timing behavior of vehicular software systems.

Another comparison framework, developed in [19], takes into consideration mainly the functional, but also non-functional aspects of modeling languages. It identifies the aspects of the software architecture that should be modeled by an architectural language. An extension of this framework is presented in [20], which incorporates several other features of hardware architectures and non-functional characteristics such as timing and dependability for safety-critical systems. Even though the framework is extended, modeling of timing information is not the primary focus of the work in [20]. Whereas, we focus on filling this gap by providing the minimal criteria for a modeling language to describe the timing behavior of a vehicular software system and enable its timing analysis.

A survey of component-based modeling languages for embedded systems and their classification framework is presented in [21]. The survey provides a minimal criteria to classify several domain-specific modeling languages. In comparison to this work, we aim at classifying and comparing modeling languages with respect to their expressiveness to support modelling of timing information and provision of timing analysis in the vehicular domain only. The work in [22] defined a set of features and capabilities that a modeling language should support to enable timing analysis. The scope of this work is limited to two specific modeling languages. In comparison, we present a generic classification framework.

To the best of our knowledge, this work represents the first attempt to categorise and compare modeling languages for vehicular software systems based on their timing expressiveness, which is responsible for enabling timing analysis of the software architectures with efficiency and precision.

III. The Classification Framework

This section presents a comprehensive framework for categorizing and comparing different modeling languages based on their expressiveness to model timing information on the vehicular software architectures. The framework is graphically illustrated in Figure. 3. The top-level entities in this framework include the timing model and timing verification.

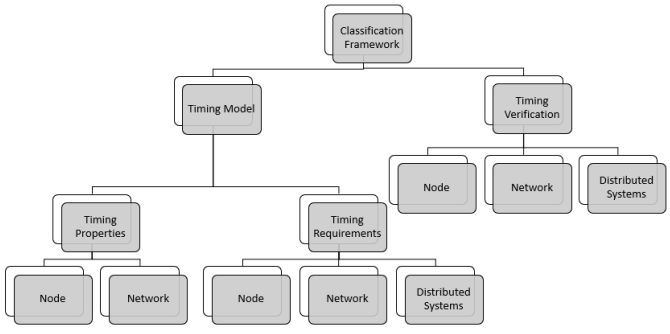

Fig. 3. Classification Framework.

A. Timing Model

The timing model is a crucial input for model-based timing analysis tools. It comprises timing properties and requirements from each node (ECU) and network in the vehicular system.

*1) **Timing Properties**:* The timing properties are categorized by node properties and network properties. The node properties are defined based on the model in [23], which can be represented by the following tuple.

$$\tau = \{C, T, D, P, J, O, B, R\} \quad (1)$$

Above, C represents the worst-case execution time (WCET) that a task takes to execute on a particular hardware without considering any interference. A task can be activated periodically or sporadically with the minimum inter-arrival time denoted by T. Moreover, a deadline D is defined as the time that the task should finish its execution in order to not disrupt the service (soft deadline) or lead to a catastrophe (hard deadline). A task has a priority which indicates the importance of the task shown by P. The period is defined either manually or using a priority assignment algorithm, such as the rate monotonic priority assignment. The time difference between the earliest and latest activation time of the task is called jitter, which is shown by J. An offset O can be set for a task which indicates the time that the task is activated after its arrival. A task can be blocked by lower priority tasks due to occupied shared resources and the maximum blocking time is denoted by B. Finally, the worst-case response time of the task R is the maximum difference between the finishing and arriving time considering the interference and blocking. Besides the above timing properties, a node in a distributed embedded system can contain several transactions with multiple tasks. Therefore, additional information related to the transactions belonging to a node should be provided. In multi-core architectures, where nodes are partitioned in two or more cores, the basic properties for nodes remain the same [24] [25]. However, additional runtime information is needed for example task affinity properties.

A distributed embedded system contains one or more networks. A network is usually characterized by a data transmission speed and the communication protocol. Similar to the tasks within a node, a message in a network has a set of properties, which are specified in the following tuple.

$$m = \{S, C, T, D, O, P, J, B, R\} \tag{2}$$

In the above tuple, a message contains a payload, which is the actual data to be transferred in the network, that is defined by S. Depending on the network speed, the message transmission time can be derived from the payload size and the message overhead, which is modeled by C. Similar to tasks, messages can be sent periodically or sporadically that is shown by T, while the deadline for the message delivery is denoted by D. A message can be activated with an offset, shown as O. The priority of the message is shown by P in the network. Moreover, the message may have a release jitter, denoted by J. The message transmission can be blocked by the lower priority message transmission that blocks the medium and the maximum blocking time is denoted by B. The worst-case response time of the message is also defined by R.

*2) **Timing Requirements**:* These requirements are specified on the software architectures by means of timing constraints on events or event chains [26]. Figure 4 exemplifies a timing requirement on the distributed functionality between the event from the sensor detecting road obstacles and the event responsible for the steering wheel actuation. Timing constraints are often specified using two attributes representing lower and upper bounds. A constraint is satisfied when the occurrences of the events happen within these limits. This framework categorizes the timing constraints into node and network constraints, which is aligned to the AUTOSAR standard [15].

The constraints that can apply on nodes are as follows.

- Delay Constraint: it constrains the time distance between the source triggering event and target response event.
- Strong Delay Constraint: this constraint is similar to the delay constraint with the difference that both source and target response events have equal period.
- Order Delay: it is a minor instance of the Strong Delay Constraint where the low attribute limit is set to zero and the high limit is set to infinite. Moreover, the occurrence of the two events are not allowed to coincide.
- Repetition Constraint: it constrains the activation pattern of the occurrences of a single event under the presence of jitter.
- Repeat Constraint: it is an instance of the Repetition Constraint where an event is not allowed to experience any Jitter.
- Sporadic Constraint: it constrains the activation pattern of two consecutive occurrences of a single event to be sporadic. Additionally, the time distance between the two occurrences must be equal or higher than the minimum inter-arrival time.
- Periodic Constraint: it is a special instance of the Sporadic Constraint where the minimum and maximum inter-arrival times are equal. This is also referred as the event's period.
- Burst Constraint: it is also a special instance of the Sporadic Constraint where Jitter is not allowed. Moreover, in a particular time-interval there is a fixed number of occurrences of the event and the time distance between two consecutive occurrences is higher than the minimum inter-arrival time.
- Pattern Constraint: it constrains the occurrences of the event to be activated following a specific pattern where fixed periodic points are pre-defined. Fixed periodic points are considered with respect to the offset.
- Arbitrary Constraint: it constrains the activation pattern of the occurrences of the event to be random and irregular.
- Execution Time Constraint: it constrains the execution time of a function where the preemptions or blocking during execution are not considered.
- Synchronization Constraint: it constrains a group of event occurrences in which an occurrence should be within a margin limit, known as the tolerance window.
- Strong Synchronization Constraint: it is similar to the Synchronization Constraint, however, the tolerance windows should not overlap.
- Output Synchronization Constraint: it constrains the occurrence of a group of events corresponding to a stimulus.
- Input Synchronization Constraint: it constrains the way how occurrences of stimuli events follow each other corresponding to a response event.
- Reaction Constraint: it constrains the time distance that a response event corresponding to a stimuli event must occur.
- Age Constraint: it is defined as the maximum data age allowed from the input to the output of a chain.

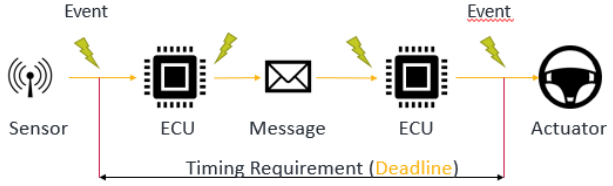

Fig. 4. Example of a timing requirement on distributed vehicular functionality.

B. Timing Verification

After the system has been modeled, its timing information should be extracted in order to provide evidences on the system timing predictability. To this end, timing analysis are used to check whether or not the constraints are satisfied. We categorize the timing verification parameters as follows:

- Verification of timing constraints applied to the frequency and occurrence of a single event such as Repetition, Repeat, Period constraints.
- Verification of synchronization timing constraints applied to a group of events.
- Verification of timing constraints applied to the time distance between the occurrence of the source triggering event and the occurrence of target response event.
- Verification of timing constraints applied to event chains or distributed event chains.

IV. Case Study

To demonstrate the applicability of the proposed framework, we use it to categorise and compare two industrial modelling languages: RCM [10] and Amalthea [11].

A. Evaluation based on Modeling of Timing Properties

Both languages have been developed for supporting the modeling of timing information of automotive software. Besides, they have same level of expressiveness for modeling node-level timing properties (Table I). The languages use different levels of abstraction. For instance, in RCM timing properties are linked to a software component. Whereas, in Amalthea these properties are liked to a Runnable, which is a schedulable part of a software component. Furthermore, a Runnable can be mapped to more than one task at run-time unlike RCM which supports a one-to-one mapping between software components and tasks.

TABLE I
TIMING PROPERTIES OF ELEMENT RESPONSIBLE FOR RUN-TIME BEHAVIOR AND COMMUNICATION.

Timing Properties					
Node	RCM	Amalthea	Network	RCM	Amalthea
WCET (C)	✓	✓	WCTT	✓	–
Priority (P)	✓	✓	Priority (P)	✓	–
Offset (O)	✓	✓	Offset (O)	✓	–
Deadline (D)	✓	✓	Deadline (D)	✓	–
Period (T)	✓	✓	Period (T)	✓	–
Jitter (J)	✓	✓	Jitter (J)	✓	–
Response Time (R)	✓	✓	Response Time (R)	✓	–
			Pattern (F)	✓	–
			Payload (S)	✓	–
			Inter-arrival Time (IAT)	✓	–

In both languages, the basic components can be part of transactions (chains), which can be event- or data-triggered. The languages support single-rate and multi-rate data chains. Both languages provide a clear separation between the control and data flow [12] [11]. Both languages support modeling of multi-core applications from the perspective of proof-of-concept prototypes. However, Amalthea has more practical support for modeling multi-core vehicular software systems.

Table I summarizes the comparison among the two languages with respect to modelling of network timing properties. RCM supports the modeling of all properties that are identified in the proposed framework. Amalthea does not support modeling of network-level timing properties. This is because Amalthea focuses only on the optimization of applications developed for single-node multi-core software architectures [13]. Since Amalthea is foreseen as a complementary language to AUTOSAR to support modeling of multi-core systems, it can be argued that Amalthea may get network modeling support indirectly via AUTOSAR. RCM is capable of supporting the development of distributed vehicular software systems and makes a distinction between intra- and inter-node communication [12]. Inter-node communication is made through the use of messages and hence expressing the properties identified in Table 2. The identified properties are applicable to several on-board networks, e.g., Controller Area Network and TSN [27].

TABLE II
TIMING CONSTRAINTS ON A SINGLE EVENT, PAIR OF EVENTS AND CHAIN OF EVENTS.

Timing Requirements		
	RCM	**Amalthea**
Single Event		
Repetition Constraint	✓	✓
Repeat Constraint	✓	✓
Sporadic Constraint	✓	✓
Periodic Constraint	✓	✓
Burst Constraint	✓	✓
Pattern Constraint	✓	✓
Arbitrary Constraint	✓	✓
Pair of events		
Delay Constraint	✓	✓
Strong Delay Constraint	✓	✓
Order Constraint	✓	✓
Set of events		
Sync Constraint	✓	✓
Strong Sync Constraint	✓	✓
Output Constraint	✓	✓
Input Constraint	✓	✓

B. Evaluation based on Modeling of Timing Requirements

This section performs a comparative evaluation of the two languages with respect to their expressiveness to model the timing requirements. These requirements are extracted from the TADL2 languages that provides a timing model for AUTOSAR. Since Amalthea inherits most of the timing constraints for events, event sets and event chains exactly from TADL2, it supports modeling of all the identified timing constraints as shown in Table II. Similarly, RCM is also capable of modeling these timing constraint [28]. Both languages are also capable of constraining all the activation patterns of a single event occurrences. They permit the activation of events periodically, sporadically, or following a certain pattern.

In Table III, we identify the timing constraints that can be specified on a chain of events and distributed chain of events. Moreover, we also identify the timing constraints, which could potentially be put on the network. RCM is capable of supporting all the timing constraints on the network as well as on the event chains and distributed event chains. On the other hand, Amalthea can model the timing constrains within a node but does not support modeling of these constraints on network

messages and distributed event chains as shown in Table III. However, how the languages specifies the supported timing constraints differs. For example, RCM provides two special objects for specifying the Delay and Strong Delay Constraints. Whereas, the Strong Delay Constraint in Amalthea can be derived from the Delay Contraint by configuring *Mapping Type* option to OneToOne [13] [28]. Besides timing constraints, Amalthea provides additional support for mapping of constraints to Runnables and Tasks to cores.

TABLE III
TIMING CONSTRAINTS ON NETWORK AND EVENT CHAINS.

Timing Requirements		
	RCM	**Amalthea**
Event chains		
Reaction Constraint	✓	✓
Age Constraint	✓	✓
Input Sync Constraint	✓	✓
Output Sync Constraint	✓	✓
Distributed event chains		
Reaction Constraint	✓	–
Age Constraint	✓	–
Input Sync Constraint	✓	–
Output Sync Constraint	✓	–
Network		
Repetition Constraint	✓	–
Periodic Constraint	✓	–
Sporadic Constraint	✓	–
Pattern Constraint	✓	–
Strong Delay Constraint	✓	–
Transmission Time Constraint	✓	–

C. Evaluation based on Support for Timing Verification

This subsection performs a comparative evaluation of RCM and Amelthea based on the tool support for timing verification of vehicular software architectures. The results of the evaluation are shown in Table IV. The timing analysis framework in Rubus-ICE is complemented by a predictable run-time environment and is supported by the Rubus real-time operating system (RTOS). The repetition constraint is proven by construction as RCM provides special clock and event objects, supported by the Rubus runtime framework, for predictable repetition rates. Similarly, the runtime framework of RCM ensures the verification of all types of single event occurrence patterns. Furthermore, the execution time constraint is verified by construction as Rubus RTOS does not allow a particular task to over-run than the specified WCET. RCM supports verification of all timing constraint on messages, event chains and distributed event chains. The input synchronization constraints in RCM are enforced by the the offline scheduler that ensures that multiple inputs to one or more SWCs are synchronized [28]. On the other hand, the verification of the output synchronization, delay, strong delay, age, and reaction constraints are supported by the end-to-end timing analysis framework of RCM.

On the contrary, APP4MC is an open-source development platform that aims to have open interfaces to create extensions for other tools regardless they are of the same nature or commercial [29]. Today, APP4MC covers only the design of real-time multi-core vehicular embedded systems and provides the run-time framework for the mapping of software components and runnable to tasks, tasks to cores, and schedulers to cores. The verification of the timing requirements needs to be performed by third-party tools as APP4MC lacks a tool for the validation of timing requirements. However, commercial tools do not expose many details regarding the modeling techniques or timing analysis that they enable. In contrast, Rubus-ICE, despite being a commercial tool, exposes the underlying analysis techniques and methods to the research community [12] [28] [30]. Some of the timing analysis tools supporting Amalthea models and APP4MC include Timing Architects [31], Symta/s [32], ChronSIM [33], MAST [34]. The first three tools do not clearly discuss the implemented timing analysis techniques and related models. Whereas, the MAST is an open-source academic tool that is transparent about the implemented techniques and analyses for single-core, multi-core and distributed systems. When MAST is used to analyse Amalthea models then a model transformation is needed for MAST to interpret the timing information modeled by MAST. One challenge with this model transformation is that some of the information may not be properly transformed because MAST does not include corresponding components for covering all aspects of the software that Amalthea supports. These challenges are pointed out in [35]. Consequently, the timing analysis engines supported by MAST may use some pessimistic assumptions about the missing information.

TABLE IV
TIMING VERIFICATION FOR EVENTS, EVENT CHAINS OR EVENTS SET.

Timing Verification		
	Rubus-ICE	**APP4MC**
The occurrence of a single event	✓	✓
The occurrence of a message	✓	–
Time distance between the occurrence of triggering event and response event	✓	✓
Synchronization Constraints on events set	✓	✓
Constraints put on event chains	✓	✓
Constraints put on distributed event chains	✓	–

V. CONCLUSION AND FUTURE WORK

In this paper, we presented a framework for capturing the timing properties and requirements that modeling languages should express to verify the timing predictability of vehicular software systems. Interpretation of timing models generated through the specification of timing properties is needed to make use of such models. Hence, the framework also targets to what extent existing tools support the interpretation and verification of timing models. We validated the applicability of the framework by considering two vehicular modeling languages, namely Amalthea and RCM.

Based on the evaluation results, we identify that both modeling languages support the design and analysis of vehicle software. However, they have different scopes. Amalthea targets applications on single nodes with multi-core architectures while RCM focuses on single-core single-node and distributed embedded systems with ongoing work for supporting single-node multi-core architectures. Both modeling languages support time-, event- and data-driven activation of software components. Furthermore, both languages support modeling of single- and multi-rate transactions. In comparison to Amalthea, RCM provides a generic message model which can easily be re-modeled according to protocol-specific properties.

Our comparative evaluation also indicates that the modeling effort and learning curve of RCM and Rubus-ICE is minimal

due to the approach of abstracting low-level implementation details and explicit modeling of timing properties, e.g., jitter. Amalthea model relies on sub-models for each aspect of software architecture, which increases its understandability and usability. APP4MC (the Amalthea tool) does not offer a timing analysis engine. Hence, it relies on third party tools for the purpose of performing timing analysis of software architectures. Amalthea is characterized by a high degree of timing expressiveness regarding single-node vehicular software systems of different architectures. However, this comes with the drawback of an extensive modelling effort. This may also result in increasing the pessimism of timing analysis as assumptions need to be made when some timing properties are not available. On the other hand, Rubus-ICE is has been upgraded continuously through several projects in collaboration with academia and industry. New components are added to adopt state-of-the-art research on timing analysis proposed by academia and to meet the needs of the end-users.

One possible future research direction is to extend and refine the proposed framework by considering the timing properties of software architectures of computation-demanding vehicular applications that would run on many-core and heterogeneous computing platforms. Another possible direction is the application of the proposed framework to additional modelling languages for automotive systems such as EAST-ADL.

ACKNOWLEDGEMENTS

The work in this paper is supported by the Swedish Governmental Agency for Innovation Systems (VINNOVA) via the PANORAMA, DESTINE, PROVIDENT and INTERCONNECT projects, and the Swedish Knowledge Foundation via the FIESTA, HERO and DPAC projects. We thank our industrial partners, especially Arcticus Systems, Volvo CE and HIAB.

REFERENCES

[1] C. Ebert and J. Favaro, "Automotive software," *IEEE Software*, vol. 34, no. 3, pp. 33–39, May 2017.

[2] A. Bucaioni and P. Pelliccione, "Technical architectures for automotive systems," in *2020 IEEE International Conference on Software Architecture (ICSA)*. IEEE, 2020, pp. 46–57.

[3] Land Rover Newsroom. https://media.jaguarlandrover.com/news/2019/04/jaguar-land-rover-finds-teenagers-writing-code-self-driving-future.

[4] L. Lo Bello, R. Mariani, S. Mubeen, and S. Saponara, "Recent advances and trends in on-board embedded and networked automotive systems," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 2, 2019.

[5] A. Bucaioni, S. Mubeen, F. Ciccozzi, A. Cicchetti, and M. Sjödin, "Modelling multi-criticality vehicular software systems: evolution of an industrial component model," *Software and Systems Modeling*, vol. 19, no. 5, pp. 1283–1302, 2020.

[6] A. Bucaioni, A. Cicchetti, and M. Sjödin, "Towards a metamodel for the rubus component model." in *ModComp@ MoDELS*. Citeseer, 2014, pp. 46–56.

[7] R. Eramo and A. Bucaioni, "Understanding bidirectional transformations with tggs and jtl," *Electronic Communications of the EASST*, vol. 57, 2013.

[8] S. Mubeen, E. Lisova, and A. V. Feljan, "Timing predictability and security in safety-critical industrial cyber-physical systems: A position paper," *Applied Sciences–Special Issue "Emerging Paradigms and Architectures for Industry 4.0 Applications"*, vol. 10, no. 3125, pp. 1–17, April 2020.

[9] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte, "End-to-end timing analysis of cause-effect chains in automotive embedded systems," *Journal of Systems Architecture*, vol. 80, pp. 104 – 113, 2017.

[10] S. Mubeen, H. B. Lawson, J. Lundbäck, M. Gålnander, and K. Lundbäck, "Provisioning of predictable embedded software in the vehicle industry: The rubus approach," in *2017 IEEE/ACM 4th International Workshop on Software Engineering Research and Industrial Practice (SER IP)*, 2017.

[11] Amalthea: Deliverable: D 3.4 Development of Scheduling Analysis and Partitioning/Mapping Support Tools. Work package 3. April 2014.

[12] 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 System journal*, vol. 10, pp. 453–482, 2013.

[13] App4mc project. help documentation (june 2020). Available:https://www.eclipse.org/app4mc/help/app4mc-0.9.8/index.html.

[14] Amalthea & Amalthea4Public ITEA3 Projects. https://itea3.org/project/success-story/amalthea-and-amalthea4public-success-story.html.

[15] AUTOSAR standard V2.2.1. https://www.autosar.org/fileadmin/user_upload/standards/classic/3-0/AUTOSAR_TechnicalOverview.pdf.

[16] R. Höttger, U. Jahn, P. Närdemann, and P. Heisig, "Teaching distributed and parallel systems with app4mc," in *International Symposium on Embedded Systems and Trends in Teaching Engineering*, 2016.

[17] S. Mubeen and T. Nolte, "On timing analysis of component-based vehicular distributed embedded systems at various abstraction levels," in *Federated Conference on Component-Based Software Engineering and Software Architecture (CompArch)*. IEEE, April 2016, pp. 277–278.

[18] K. Evensen and K. Weiss, "A comparison and evaluation of real-time software systems modeling languages," in *AIAA Infotech@Aerospace 2010*, April 2010.

[19] N. Medvidovic and R. N. Taylor, "A classification and comparison framework for software architecture description languages," *IEEE Transactions on Software Engineering*, vol. 26, no. 1, pp. 70–93, Jan 2000.

[20] A. Johnsen and K. Lundqvist, "Developing Dependable Software-Intensive Systems: AADL vs. EAST-ADL," in *Reliable Software Technologies - Ada-Europe*, 2011.

[21] I. Crnkovic, S. Sentilles, A. Vulgarakis, and M. R. V. Chaudron, "A classification framework for software component models," *IEEE Transactions on Software Engineering*, vol. 37, no. 5, pp. 593–615, 2011.

[22] S. Anssi, S. Gérard, S. Kuntz, and F. Terrier, "AUTOSAR vs. MARTE for Enabling Timing Analysis of Automotive Applications," in *SDL: Integrating System and Software Modeling*, 2012.

[23] K. Tindell, "Adding time-offsets to schedulability analysis," in *Technical Report YCS 221, Dept. of Computer Science, University of York.*, 1994.

[24] S. Mubeen, M. Gålnander, J. Lundbäck, and K.-L. Lundbäck, "Extracting timing models from component-based multi-criticality vehicular embedded systems," in *15th International Conference on Information Technology : New Generations*, April 2018.

[25] M. Bertogna and M. Cirinei, "Response-time analysis for globally scheduled symmetric multiprocessor platforms," *Proceedings - Real-Time Systems Symposium*, pp. 149–160, January 2008.

[26] TIMMO Methodology , Version 2. TIMMO (TIMing MOdel), Deliverable 7, October 2009. The TIMMO Consortium. [Online]. Available: http://adt.cs.upb.de/timmo-2-use/timmo/pdf/D7_TIMMO_Methodology_Version_2_v10.pdf,Accessed: 2020-06-09.

[27] S. Mubeen, M. Ashjaei, and 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)*, 2019, pp. 131–139.

[28] S. Mubeen, T. Nolte, M. Sjödin, J. Lundbäck, and K.-L. Lundbäck, "Supporting timing analysis of vehicular embedded systems through the refinement of timing constraints," *International Journal on Software and Systems Modeling*, pp. 1–31, January 2017.

[29] R. Höttger, H. Mackamul, A. Sailer, J.-P. Steghöfer, and J. Tessmer, "APP4MC: Application platform project for multi- and many-core systems," *it - Information Technology*, vol. 59, no. 5, pp. 243 – 251, 2017. [Online]. Available: https://www.degruyter.com/view/journals/itit/59/5/article-p243.xml

[30] M. Ashjaei, S. Mubeen, J. Lundbäck, M. Gålnander, K. Lundbäck, and T. Nolte, "Modeling and timing analysis of vehicle functions distributed over switched ethernet," in *IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society*, Oct 2017, pp. 8419–8424.

[31] "Vector - Timing Architects Tool Suite". [Online]. Available: https://www.timing-architects.com/, Accessed: 2020-06-10.

[32] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, "System level performance analysis - the symta/s approach," *IEE Proceedings - Computers and Digital Techniques*, vol. 152, no. 2, 2005.

[33] "chronSIM, Model-Based Simulation of Embedded Real-Time Systems". https://www.inchron.com/tool-suite/chronsim/, Accessed: 2020-06-10.

[34] ”MAST-Modeling and Analysis Suite for Real Time Applications”. [Online]. Available: https://mast.unican.es/, Accessed: 2020-06-09.
[35] J. M. Rivas Concepción, J. J. Gutiérrez, J. Medina, and M. Harbour, “Calculating latencies in an engine management system using response time analysis with mast,” in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 07 2018.