Problem-Based Learning in an Educational and Training Module on Model-Based Development of Vehicle Software

Saad Mubeen
saad.mubeen@mdu.se
Mälardalen University
Västerås, Sweden

Mohammad Ashjaei
mohammad.ashjaei@mdu.se
Mälardalen University
Västerås, Sweden

Abstract

This paper presents the development and evolution of an educational module centered on collaborative problem-based learning, model-based software development, and end-to-end timing analysis for vehicular embedded systems, highlighting our experiences in integrating state-of-the-art research and industry practices over the years. For the past eight years, the module has been taught to both industry professionals and academic students. In the industry, it has been presented through seminars and workshops organized within the industrial settings of two vehicle manufacturers and a provider of vehicular software development tools. In an academic context, this module has been delivered as part of a PhD course. Furthermore, it has been incorporated into 11 instances of master's courses across four European universities. When offered in an industry context, the module is kept concise and more focused on hands-on activities and practical use cases. In contrast, when the module is delivered in an academic setting, it is supplemented with additional lectures and discussions on its topics. Interestingly, the feedback received from participants, especially those from the industry, has not only contributed to refining this educational module but has also advanced the state of the art in modeling and timing analysis of embedded software architectures.

Keywords

Automotive software, model-based development, timing analysis.

ACM Reference Format:
Saad Mubeen and Mohammad Ashjaei. 2025. Problem-Based Learning in an Educational and Training Module on Model-Based Development of Vehicle Software. In *Proceedings of The 29th International Conference on Evaluation and Assessment in Software Engineering (EASE 2025).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

The size and complexity of software running on onboard embedded computers, known as Electronic Control Units (ECUs), in modern vehicles have been increasing tremendously over the past few years [20]. The software size has already reached the order of 100 million lines of code [6, 29]. According to a study by Jaguar Land Rover [5], the size of vehicle software is projected to reach 1 billion lines of code soon. This drastic increase in software size and complexity is driven by the increasing demand for new software-based features in vehicles. These features often require advanced high data-rate sensors (e.g. cameras and lidars) and next-generation high-bandwidth and low-latency onboard communication, such as Time Sensitive Networking (TSN) [1, 7]. Consequently, developing vehicle software has become a daunting task. Additionally, many vehicular software functions are constrained by various timing requirements. Therefore, vehicle software developers must not only manage software complexity but also analyze and verify the specified timing requirements during development.

Model-based software development approaches that employ the principles of Model-based Engineering (MBE) and Component-based Software Engineering (CBSE) [25, 34], along with real-time scheduling and schedulability analysis [9, 12, 31], have proven effective in addressing the aforementioned challenges in vehicle software development within the industry [20]. Therefore, it is crucial to incorporate these approaches into educational modules for software engineering students, particularly those specializing in embedded software systems and automotive software engineering programs. These educational modules can also benefit doctoral students conducting research in this field. In addition, these modules can be valuable in training programs within the vehicle industry, especially when they are updated and refined with the latest state-of-the-art results.

This paper presents an educational and training module that integrates collaborative Problem-based Learning (PBL) with a multi-phase model-based development approach for vehicular embedded software systems. The presented approach is based on several academic and industrial embedded software development approaches that are used in the vehicular domain. The module includes several industrial use cases. A key component of the module is collaborative PBL, where participants initially learn to construct the problem, enabling them to generate knowledge based on their understanding, and subsequently refine and solve it collaboratively. The module also includes a demonstration of the Rubus-ICE[1] [22] industrial tool chain used to solve the same problem, followed by a discussion on the comparative evaluation of the solutions. Furthermore, the paper discusses our experience of how the module and included approach are refined over the years based on the evolving industrial needs of next-generation vehicular systems and advancement in the state-of-the-art research to meet those needs.

This module has been taught as part of a PhD course at Mälardalen University (Sweden) and has also been included in 11 instances of master courses. Specifically, it has been part of 7 instances in

[1] https://www.arcticus-systems.com/products/rubus-tool-suite

EASE 2025, Istanbul, Türkiye

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

a course that is included in three Master programs at Mälardalen University: (1) Master Program in Intelligent Embedded Systems[2], (2) Master Program in Robotics[3], and (3) Master Program in Dependable Systems[4]. Additionally, it has been taught in 2 instances at J. J. Strossmayer University of Osijek (Croatia), 1 instance at Technical University of Eindhoven (The Netherlands), and 1 instance at University of L'Aquila (Italy). Over the past 8 years, the module has also been presented to two vehicle manufacturers and one vehicular software development tools provider through several seminars and workshops. Note that when the module is delivered in an academic setting, it is supplemented with additional lectures and discussions on its topics. In contrast, when offered in an industry context, the educational module is kept concise and more focused on hands-on activities and practical use cases. Feedback on requirements and use cases from the industry has contributed not only to the refinement of the model-based software development approach in this module but also to the advancement of the state of the art in modeling and timing analysis of embedded software architectures.

The rest of the paper is organized as follows. Section 2 presents an introduction to the educational and training module. Section 3 presents the model-based software development approach. Section 4 presents industrial relevance of the approach. Section 5 discusses the module refinement based on the industry feedback and advances in the state of the art. Section 6 concludes the paper.

2 PBL-based Educational and Training Module

The educational and training module spans the areas of model-based engineering, component-based software engineering, and real-time systems. The module offers in-depth theoretical knowledge and state-of-the-art techniques for the development and timing verification of advanced embedded software systems. The module has a particular focus on the vehicular domain.

When offered in academia, the module aims to achieve two main goals: (i) to prepare participants for PhD studies by providing a solid theoretical foundation, and (ii) to equip participants for careers as embedded software engineers in the industry by introducing them to various technologies used in the development of modern embedded software systems. The participants receive both formative and summative assessments on their individual and collaborative problem-based learning assignments. When the module is part of a PhD course, it is delivered as a two-day workshop. Conversely, when it is included in a master's course, the module extends over 4 weeks, featuring 8 three-hour lectures, take-home assignments, and a half-day collaborative workshop.

When offered in industry, the module aims to provide a hands-on tutorial for the Rubus-ICE model-based software development tool chain. It enables participants to effectively utilize the tool chain and techniques for developing vehicle software. In addition, it deepens their understanding of the timing analysis framework supported by the tool chain. This allows the participants to accurately analyze the timing behavior of vehicle software architectures in industrial applications. This module can be offered as a one- to two-day training and discussion workshop.

The module consists of five components. These components are discussed as follows.

1 – Lectures and Interactive Discussions
The first component includes a set of lectures and interactive discussions. These explain the context, motivation, academic and industrial relevance, and details of the model-based development approach for vehicular embedded software systems. Highlights of this component will be discussed in detail in Section 3.

2 – PBL Assignment: Construct Your Own Problem
This component facilitates the participants to build new knowledge with the help of problem-based learning (PBL) [2]. PBL is an educational approach where participants learn by engaging with and solving real-world problems. The purpose is that the participants construct their knowledge about the theory, concepts, and techniques covered in the first component by their practical application on problems. This method encourages active learning, critical thinking, and the ability to apply theoretical knowledge to practical situations. The assignment in this component requires each participant to construct their own comprehensive problem. They do this by investigating the basic requirements and rudimentary information provided to them and coming up with concrete assumptions and comprehensive information. The problem is then solved both individually and collaboratively in the later components.

A highlight of the assignment is depicted in Fig. 1, where the participants are provided primitive software architecture of a simplified steer-by-wire use case [28]. Participants are tasked with verifying the predictability of the software architecture with respect to its end-to-end timing. They need to identify the necessary information required for this analysis and determine what additional details are needed in the software architecture to perform a thorough end-to-end timing analysis. Furthermore, they must explore the methods and techniques for conducting timing analysis on the software architecture.

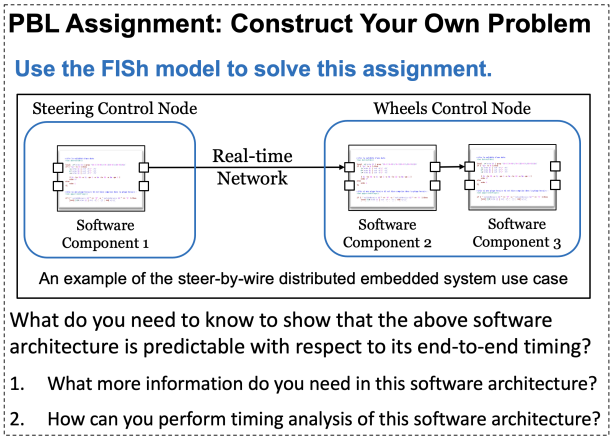

Figure 1: PBL Assignment: Construct your own problem.

3 – Collaborative PBL Assignments
The participants engage in solving the constructed problem in the previous component through a combination of independent investigation and collaborative PBL group investigation. We experimented

[2]https://www.mdu.se/en/malardalen-university/education/international/programme/masters-programme-in-intelligent-embedded-systems
[3]https://www.mdu.se/en/malardalen-university/education/programme-syllabus?id=1114
[4]https://www.mdu.se/en/malardalen-university/education/programme-syllabus?id=1740

with various group sizes, ranging from 2 to 6 individuals per group. From our experience and the results of the PBL discussions, we found that the most effective collaboration occurred with groups of 5 persons in master education, 3 persons in PhD education, and 2 persons in industry training.

3 (a) – Collaborative PBL using the FISh Model

We encourage the participants to use the FISh (Focus, Investigate and Share) model [24] in both independent and group investigations. The FISh model is a useful tool to understand how to effectively approach problem-solving by encouraging participants to reflect on their perceptions, investigate answers individually and collaboratively, and share their findings to support a deeper understanding and collaborative learning environment. The constructed problem in the previous component sets the stage for exploration. In the Focus phase, participants reflect on their perceptions of the scenario, noting any questions or reflections that arise. They observe what they see, draw on their previous experiences, and identify what they already know. This phase also involves specifying a few key questions to be investigated further. In the Investigate phase, participants discuss ideas on where to find answers to their questions and decide on how to proceed both individually and as a group. They individually study the suggested resources or conduct their own searches, summarizing their findings. Finally, in the Share phase, participants come together to share their insights and reflections with the group, facilitating a collaborative learning environment.

3 (b)–Collaborative PBL using the Iterative Fork-Join Model

Effective collaboration in a PBL group requires continuous collaboration within the group as opposed to the case where group members investigate the problem individually in isolation and finally put together the investigation results. No doubt, it is important that the members perform investigation individually in their own time. However, the individual investigations need to be discussed and refined time and again in achieving the intended results. To this end, we developed the iterative fork-join model for collaborative problem solving in each PBL group and encouraged the participants to apply this model. In this model, shown in Fig. 2, each individual member or subgroup breaks out to investigate the problem independently in the first iteration. At the end of the iteration, the group members present their individual findings to each other and discuss them for the purpose of refining the findings, investigation inputs and other related parameters. The iterations are repeated several times to achieve more refined results. The participants utilize the FISh model within the iterative fork-join model to solve collaborative PBL assignments. Note that each PBL group solves the problem that was initially constructed in the previous component and further refined at the beginning of this component.

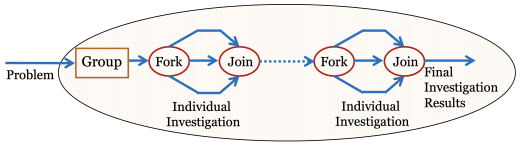

Figure 2: Iterative fork-join model for collaborative PBL.

4 – Infusing Industrial Tool Demo with Collaborative PBL

This component enhances participants' learning by incorporating industry solutions. The problem developed and refined in previous components is now modeled, timing analyzed, and synthesized using Rubus-ICE (Integrated Component model development Environment)[5] [22, 23], a tool utilized in the vehicle industry. This approach allows participants to experience solving the same problem with an industrial tool, thereby enhancing their learning by comparing their solutions with some of the industrial solutions.

5 – Concluding Discussion and Feedback

Towards the end of the module, a workshop is organized for all groups to share and discuss their results, along with the assumptions and design decisions that led to those results. This allows the groups to learn from each other's experiences. The discussion and feedback have the potential to provide valuable input for the industrial tool or contribute to advancing the state of the art in the field. Further discussion on this will be presented in Section 5.

3 Model-based Software Development Approach

The participants of this module are presented with a generic model- and component-based software development approach for vehicular embedded systems, as illustrated in Fig 3. This approach is organized in various phases, where each phase represents the complete vehicular embedded software system for a given purpose: requirements elicitation, software architecture modeling, analysis, synthesis, simulation and testing, and deployment and execution. Note that we do not cover business, organizational and post-deployment operational aspects in this approach. This approach is inspired by several academic and industrial embedded software development approaches in the vehicular domain, such as the approaches used by EAST-ADL modeling language [3, 19], AUTOSAR standard [13, 32], ISO26262 Standard [16], Fraunhofer ESK [4], ProCom component model [30], COMDES-II component model [18], and Rubus Component Model (RCM) [11, 14, 22], to mention a few.

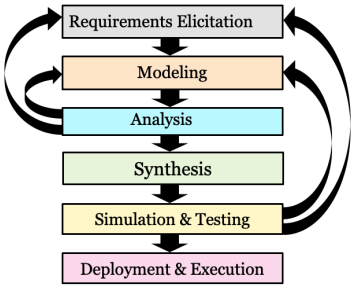

Figure 3: A typical approach for model-and component-based development of vehicular software architectures.

3.1 Requirements Elicitation

The first phase in this development approach is to identify broad needs of the customer and capture end-to-end requirements on the vehicle functionality. Furthermore, requirements on the use cases and proof-of-concept prototypes may also be identified in this phase. The requirements are often captured in an informal way using natural language. There are several tools that support requirement elicitation for vehicle software, e.g., SystemWeaver[6] and CATIA No Magic[7].

[5] https://www.arcticus-systems.com/products/rubus-tool-suite
[6] https://systemweaver.com
[7] https://www.3ds.com/products-services/catia/products/no-magic/

In this educational module, we mainly focus on timing requirements (e.g., response times, jitter, age delays and reaction delays) and other resource requirements (e.g. computational power, memories and bus/network bandwidth). Timing requirements are often derived from functional requirements, design choices, and environmental and physical limitations. For example, the timing requirement on a decision to deploy an airbag is 15-30 milliseconds. Similarly, the timing requirement on inflating the airbags is 50-80 milliseconds after the onset of the crash.

3.2 Software Architecture Modeling

In this phase, models of software architectures of distributed vehicular embedded systems are developed. The software architecture model comprises of software components (SWCs) representing software functions, interconnections among the SWCs, structures and other design artifacts. An SWC is the lowest-level hierarchical element in the software architecture. An SWC can have one or more behaviors. For example, an SWC can have a separate behavior in each of start-up, running, stand-by and low-power modes.

In addition to modeling the software architecture in this educational module, we focus on modeling timing properties and requirements on the software architectures. Some examples of these properties include activation sources of SWCs (e.g., periodic clocks and sporadic events), periods of periodic clocks, minimum inter-arrival times of event sources, priorities of SWCs, precedence relations among SWCs (an SWC must finish its execution before starting the execution of subsequent SWC), and Worst Case Execution Time (WCET) of the SWC representing the maximum time it takes for the SWC to execute from start to end in isolation.

An example of a software architecture modeled with EAST-ADL modeling language is presented in Fig. 4. This software architecture consists of four SWCs. The SWCs interact with each other through flow ports that are identified as data input and data output ports in Fig. 4. Each SWC has a name (e.g., C_1), an entry function or behavior (e.g., F_1) and worst-case execution time (e.g., $WCET_1$).

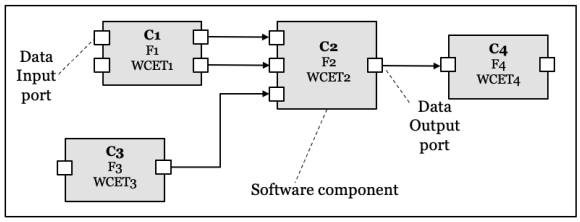

Figure 4: An example of a software architecture modeled with EAST-ADL modeling language.

Fig. 5 depicts the software architecture example of a two-node distributed vehicular system modeled with Rubus modeling language. Note that a timing constraint is specified between the models of a sensor and an actuator, while the path from the sensor to the actuator traverses through two nodes (Electronic Control Units) and one network. The internal software architecture of one of the nodes is also presented in Fig. 5. Some SWCs are triggered by periodic clocks while others are triggered by sporadic events that are generated by their predecessor SWCs. The models of network, messages and signals contained in the messages are also shown in this figure. Note that the control and data flows are clearly separated in this architecture. Such a separation supports timing analysis of the software architecture with ease and high precision [23, 28].

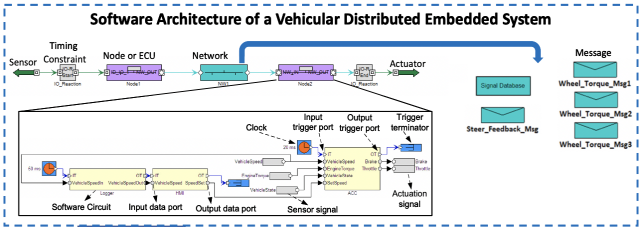

Figure 5: Software architecture example of a distributed vehicular system modeled with Rubus modeling language.

3.3 Analysis

The software architecture modeled in the previous phase can be analyzed for various purposes, e.g., analysis of refined requirements in the software architecture, consistency analysis, timing analysis, to name a few. In this educational module, we focus only on model-based timing analysis of software architectures [23, 27, 28]. These analyses include response-time analysis of SWCs, response-time analysis of network messages, and end-to-end response-time analysis as well as end-to-end data propagation delay analysis of distributed chains of SWCs and messages in the software architectures. These analysis are supported by several model-based timing analysis tools such as Rubus, VNA, Symtavision, and several tools in the AUTOSAR tool chain, to mention a few.

Response-time analysis is a classical schedulability analysis technique that calculates the worst-case response-time of each SWC (task)[8] and compares it with the corresponding timing constraint (deadline) [31]. The worst-case response time of an SWC is the sum of all the interference from higher and equal priority SWCs, blocking from lower priority SWCs and WCET of the SWC itself as depicted in Fig. 6.

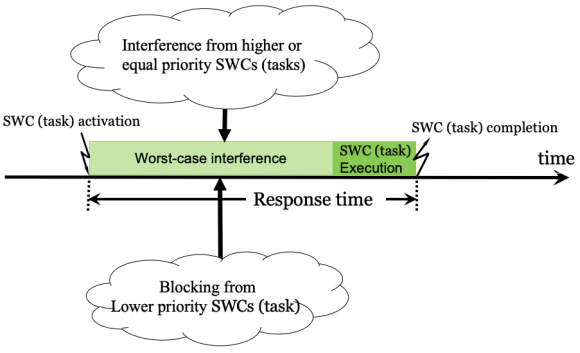

Figure 6: Components of the response time of an SWC.

The **end-to-end data propagation delay analysis** calculates the data propagation delays from the input to the output across a distributed chain of SWCs and network messages [9, 10, 12]. For example, a distributed chain is identified with an arrow in Fig. 7.

[8]Software component is a design-time entity. It may correspond to a schedulable entity at runtime, e.g., an operating system task [8].

The chain is initiated with a data input from a sensor, which is acquired by Node1. The data then propagates through the network to Node2 where it propagates through a chain of SWCs, and finally towards an actuator. There are two data-propagation delays that are commonly analyzed in the vehicular domain, namely *age* and *reaction* delays. These delays (age and reaction delays) and their corresponding constraints (age and reaction constraints) are part of the AUTOSAR Standard and several industrial modeling languages such as EAST-ADL, TADL2 and Rubus. A reaction constraint is specified from the input to the output of the chain in Fig. 7.

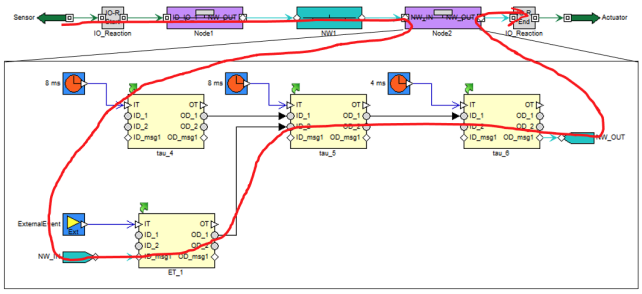

Figure 7: Visualization of end-to-end delay in a distributed embedded software architecture.

In order to explain the age and reaction delays, consider a simple example of a software architecture consisting of three SWCs as shown in Fig 8. Various timing properties (periods and priorities) of SWCs are specified in the figure. The WCET of each SWC is assumed to be 1 time unit. There can be four possible delays that the data can experience from the input to the output of the chain. Only two of these delays (age and reaction) are important in the vehicular domain [26, 32] and are identified with thick arrows in Fig 8. The age delay refers to the delay between the last input of the chain (that is not overwritten) until the last output of the chain (even in case of duplicate outputs). Whereas, the reaction delay refers to the first reaction (output) to any input event that was just missed by the initiator SWC at the input of the chain.

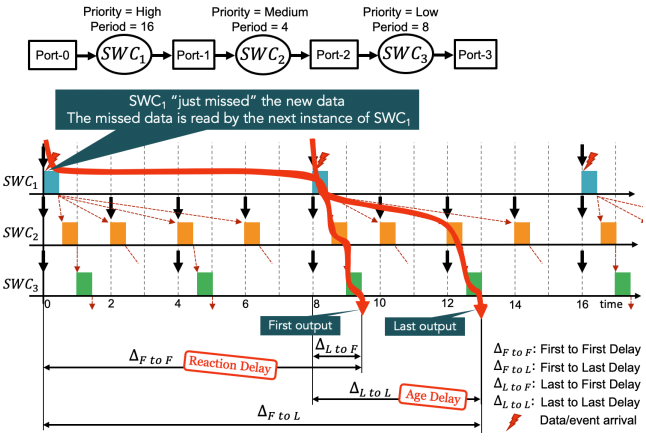

Figure 8: Various end-to-end delays in the vehicle software.

The age constraint finds its significance in the control systems domain, where maximum age of the data is of significance such that the control signals driving the actuators do not exceed the maximum age. Consider the example of a cruise control system in a car. A sudden change in the speed is required if the car goes up/down a hill. If the age of the data is too long then the car will speed up/slow down with noticeable delay, which is undesired. **The reaction constraint finds its applications in the body electronics domain** in vehicles such as the button-to-reaction functions, e.g., electronic door lock in a car. When the door-locking button is pressed, the primary focus is on the first reaction, which is locking the door, to the button input. Any subsequent reaction is of no use as an already locked door cannot be further locked.

In order to perform the above mentioned analyses, the *end-to-end timing models* need to be extracted from the software architectures and fed as input to the timing analysis engines as shown in Fig.9. An end-to-end timing model comprises timing properties and requirements on individual SWCs and messages as well as on distributed chains of SWCs and messages in the software architecture [21]. The analysis results are used to verify the timing requirements that are specified on the software architectures. If the requirements are not met, the analysis results can guide the developers to refine the software architecture or even refine the requirements as depicted by the backward arrows from the analysis phase to the modeling and requirements elicitation phases in Fig. 3.

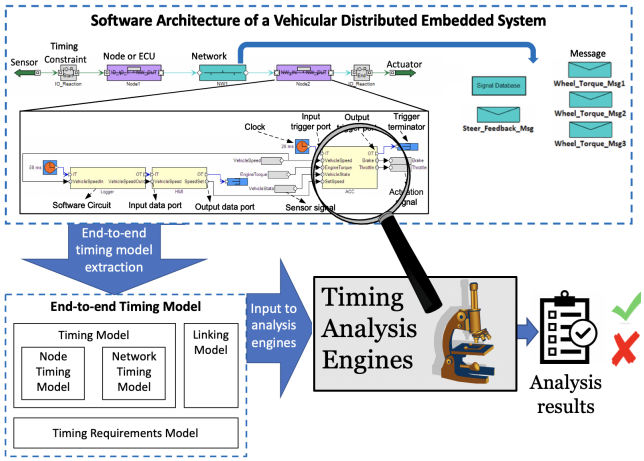

Figure 9: Extraction of end-to-end timing models from the software architectures to support their timing analysis.

3.4 Synthesis

In this phase, synthesizable code for the run-time framework is automatically generated from the timing verified software architectures. For example, a fragment of the generated code by Rubus code generation engine is depicted in Fig. 10 that shows some data structures of an SWC. Note that this is not behavioral code (representing internal logic of the SWC). Many software architecture modeling tools in the industry, e.g., Rubus, are integrated to Simulink that allows modeling of behaviors of the SWCs. In this way, the behavioral code of SWCs can also be generated automatically.

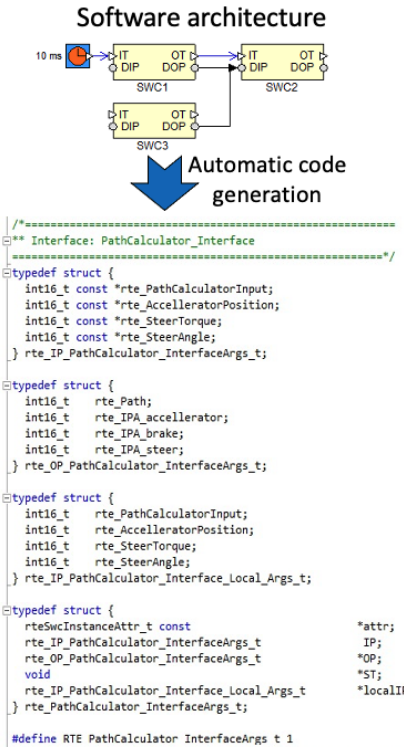

Figure 10: An example of automatically generated code from a vehicular software architecture.

3.5 Simulation and Testing

In this phase, a simulation and model-based testing [33] of the software architecture is carried out. The element that can be simulated and/or tested can be a single SWC, an assembly (a group of SWCs), software architecture within a node or electronic control unit, software architecture of entire distributed vehicular embedded system as shown in Fig. 11. Additionally, various high-level tools such as LabVIEW or Simulink can be used for feeding the simulation process with specific inputs as depicted in Fig. 12.

The simulation and testing results can also guide the developers to refine the software architecture or even refine the requirements as depicted by the backward arrows from the simulation and testing phase to the modeling and requirements elicitation phases in Fig. 3.

3.6 Deployment and Execution

In this phase, the synthesized code is deployed to both software and hardware platforms. The software platforms comprise typical real-time operating systems (RTOS) such as Rubus RTOS, MicroC OS, VxWorks, FreeRTOS, to mention a few. Whereas, the hardware platforms comprise the ECUs or processors that run the RTOS and reside inside the vehicle. Alternatively, if the hardware platforms are not available then a version of the RTOSs can be adapted to host general-purpose operating systems like Windows or Linux, which in turn, can be used as the software platform. In this case, regular PCs can act as ECUs which can be connected by various types of networks like Controller Area Network (CAN) [17] and TSN. This

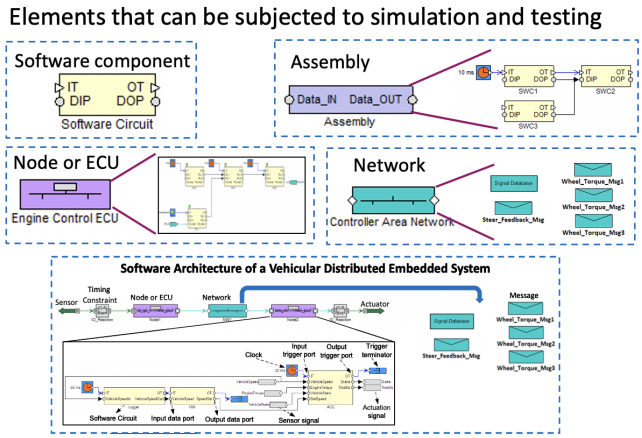

Figure 11: An example of simulation and testing environment for the software architecture.

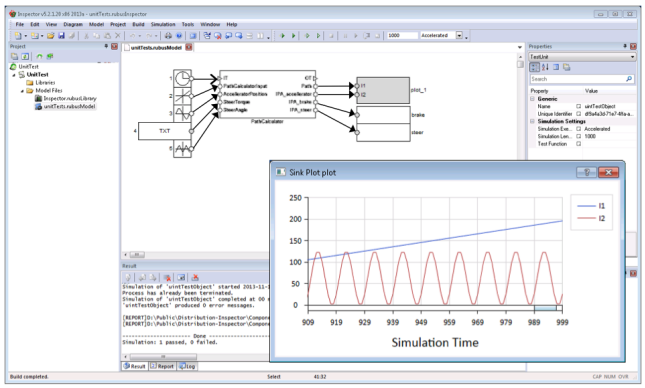

Figure 12: An example of simulation and testing environment for the software architecture.

can be useful to understand, simulate and test the system if actual ECUs are not available or are not configured yet. Multiple deployment scenarios used by the Rubus-based software development and deployment process are depicted in Fig. 13

4 Industrial Relevance of the Approach

This section presents use cases of two industrial model-based software development approaches that align with the approach discussed in the previous section. These use cases are also included as part of the educational and training module.

4.1 Example of EAST-ADL

EAST-ADL is a domain-specific architecture description language in the vehicular domain. EAST-ADL also provides a development approach for the software architecture development at four abstraction levels: vehicle, analysis, design and implementation as shown in Fig. 14. Note that the fifth level, called the operational level, is out of the scope of this module. Each abstraction level presents complete definition of the system for a specific purpose. Several

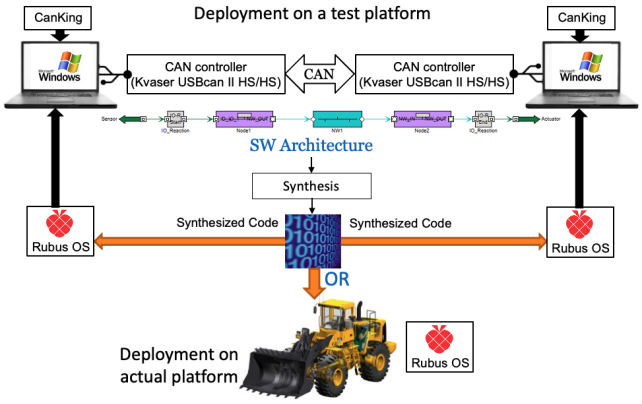

Figure 13: Various deployment scenarios of synthesized code from the software architectures.

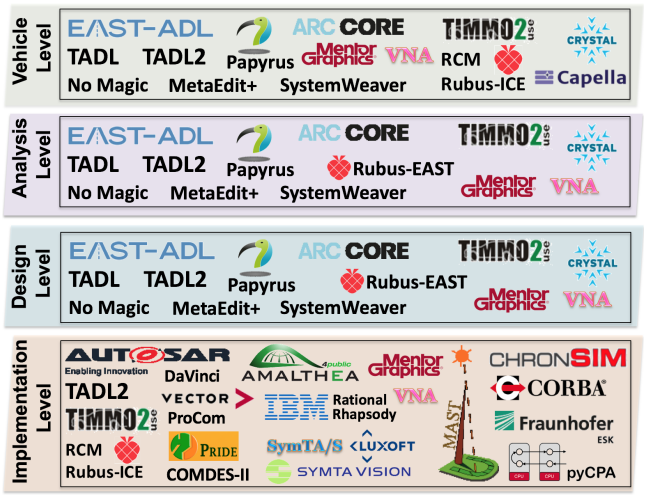

Figure 14: Examples of some industrial and academic component models and tools for the development of vehicular software architectures at various abstraction levels.

academic and industrial component models, languages and tools for the development of vehicle software are also depicted in Fig. 14. The industrial members in the EAST-ADL association that led the development and extension of the language are presented in [28].

The **vehicle level**, also referred to as the end-to-end level by some vehicle manufacturers, captures requirements on the end-to-end functionality of the vehicle. The requirements are captured in an informal (often textual) and solution-independent way. This level is aligned to the requirements elicitation phase in the approach presented in Section 3.

The **analysis level**, formally captures the requirements in an allocation-independent way. Functionality of the system is defined based on the requirements and features without implementation details. Various types of analysis can be performed at this level, including functional verification, consistency analysis, and high-level timing analysis [28]. This level is aligned to a combination of modeling and analysis phases in the approach presented in Section 3.

The **design level** describes the software architecture that is abstracted from implementation details. The refined artifacts from the analysis level include design-level software components, middleware abstraction, hardware architecture, and software functions to hardware allocation. This level corresponds to the modeling phase in the approach presented in Section 3.

The **implementation level** describes concrete implementation of the software architectures in terms of software components and their inter-connections. The software components at this level resemble black boxes that have no details about their behavior code. This level has sufficient information included in the modeled software architecture to perform its end-to-end timing analysis. This level is aligned to a combination of modeling and analysis phases in the approach presented in Section 3.

4.2 Example of Rubus

Rubus is a collection of methods, tools, a component model and an approach for model- and component-based software development of vehicular embedded systems. The Rubus tools include modeling tools, code generators, analysis tools and run-time infrastructure. Rubus also includes a real-time operating system which is certified in the ISO 26262:201110 [16] safety standard according to ASIL D. Rubus is developed by Arcticus Systems in close collaboration with several academic and industrial partners. The approach and accompanying tools have been used in the vehicle industry for over 0 years [22]. Rubus is mainly used for the development of control functionality in vehicles by several vehicle manufacturers such as Volvo Construction Equipment[9], Mecel, Knorr-Bremse, Hoerbiger, BAE Systems Hägglunds[10], to name a few. The Rubus approach to model- and component-based software development is depicted in Fig. 15. Note that all phases in the Rubus approach are perfectly aligned to all the phases except the first phase in the development approach discussed in Section 3. The Rubus approach does not focus on requirements elicitation, in fact, it relies on other tools and languages to acquire the requirements.

5 Industrial and Research Feedback in the Module Refinement

In order to demonstrate how the model-based software development approach is refined based on the industrial feedback and research results, consider the refined and annotated version of the two-node distributed vehicular embedded system modeled in the collaborative exercise in the educational module, shown in Fig. 16. The initial versions of the exercise considered only CAN network between the two nodes. CAN is an event-triggered and non-synchronized network protocol and is the most widely used traditional real-time onboard communication protocol in the vehicular domain [20]. Fig. 16 shows a model of a distributed chain of SWCs and a message. The chain is initiated by reading data from Port-0 in Node1 and terminated by providing data to Port-2 in Node2. Two timing constraints, Age and Reaction, are specified on the chain. The corresponding analysis results (calculated age and reaction delays) are shown in Fig. 17. Note that the analysis results

[9]https://www.volvoce.com
[10]http://www.baesystems.com

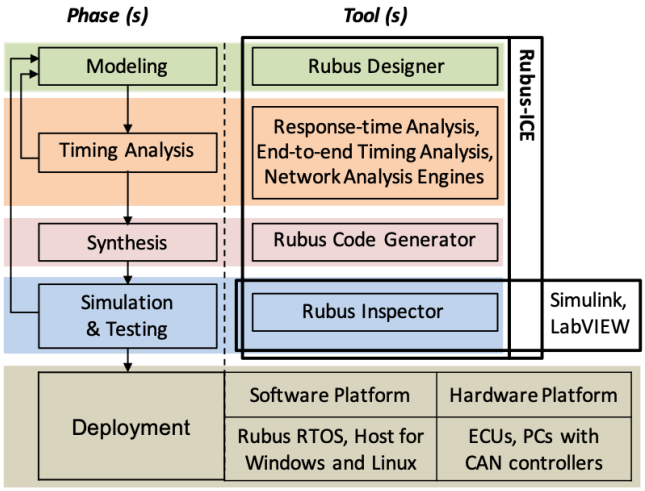

Figure 15: Rubus approach to model- and component-based development of vehicular software architectures.

are based on the assumption that any node receives a network message using the polling policy, where the network interface is periodically checked for any new message. This policy is commonly used in the automotive industry.

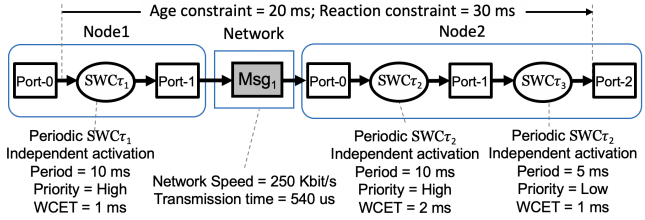

Figure 16: Refined and annotated version of the problem in collaborative PBL assignment in the educational module.

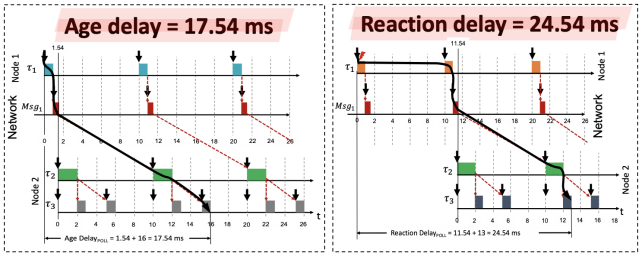

Figure 17: Calculated age and reaction delays corresponding to the constraints specified in Fig. 16.

When we presented this module in the educational seminars at several partner companies (vehicle manufacturers and tool provider), based on their feedback we realized that some vehicle manufacturers use polling policy while others use interrupt policy for receiving network messages. The analysis used in the educational module as well as the analysis implemented in the Rubus tool chain did not support the interrupt policy. This decision can make a significant impact on the timing analysis results as indicated in the top two sub-figures in Fig. 18 (different age delays) and Fig. 19 (different reaction delays). Hence, the analysis as well as the implementation in the tool was refined to support more generic use cases, where some nodes may support the polling policy while others support the interrupt policy. Moreover, this decision needs to be supported in the Rubus component model and language such that an attribute corresponding to the policy selection can be annotated to the node model within the software architecture.

Another interesting feedback received from the industry in the seminars is that the vehicular industry is exploring the possibility of utilizing the next-generation high-bandwidth and low-latency onboard communication networks like TSN which support synchronization among the nodes within a distributed embedded system. They are interested in incorporating modeling and analysis of these networks within their model-based software development approach. Once again, the modeling and analysis used in the initial version of this educational module as well as the implementations in Rubus did not support these next-generation onboard networks. As TSN supports synchronization, this can also have a considerable impact on the end-to-end delays as indicated in the bottom two sub-figures in Fig. 18 (different age delays) and Fig. 19 (different reaction delays). To support the modeling (defining new properties in the software architectures) and timing analysis of these networks within the development approach, the state of the art needed to be extended as well as the Rubus tool chain needed to be extended [15, 21].

In crux, the model-based development approach discussed in this educational and training module was refined based on the feedback from industrial partners. Moreover, previous modeling techniques needed to be extended and new timing analysis techniques needed to be developed and integrated into the software engineering environment based on the received feedback. This collaborative refinement is depicted in Fig. 20.

6 Summary and Conclusion

In this paper, we have presented the contents and shared our experiences on how an educational module focused on model-based software development and end-to-end timing analysis of vehicular embedded systems has evolved over the years. This evolution has been driven by the incorporation of state-of-the-art research and industry practices, ensuring that the module remains relevant and effective in addressing the needs of both academic and industry participants. For the past eight years, the module has been taught to both industry professionals and academic students. In the industry, it has been presented through seminars and workshops organized within the settings of two vehicle manufacturers and a provider of vehicular software development tools. In academia, this module has been part of a PhD course and included in multiple instances of master courses across various universities.

The collaborative problem-based learning approach used in this module aided by the FISh and Iterative fork-join models has yielded positive outcomes, with participants demonstrating a deeper understanding of concepts and articulating the holistic relevance of their knowledge. This is evident through positive classroom/onsite feedback, improved course evaluations, fewer revisions of individual

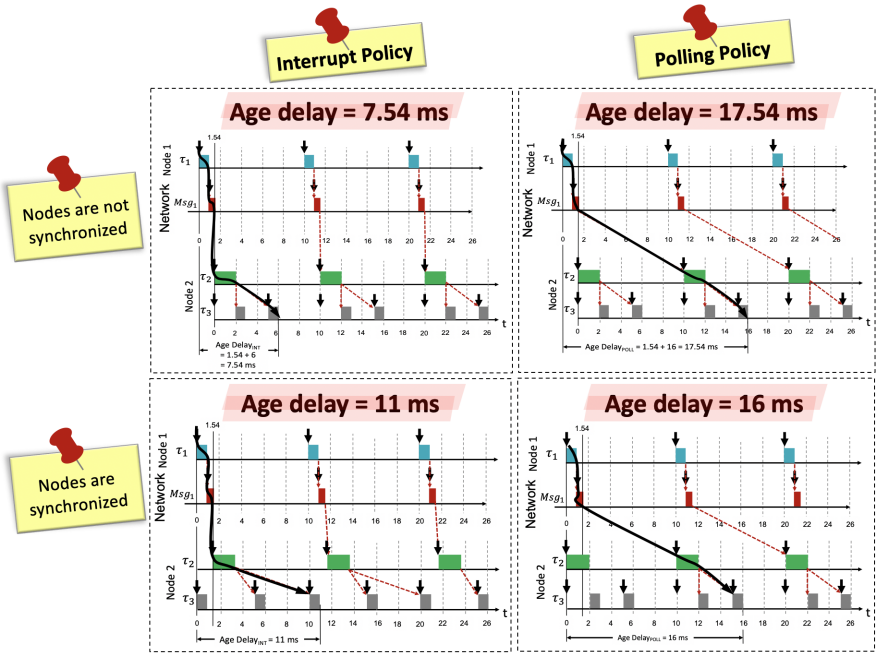

Figure 18: Results of the refined analysis based on the the participants' feedback.

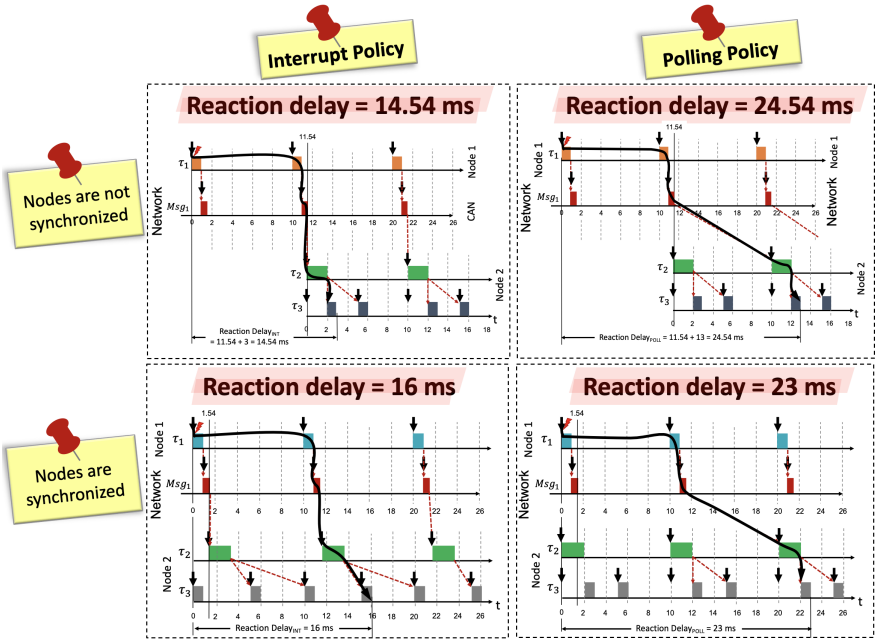

Figure 19: Results of the refined analysis based on the the participants' feedback.

assignments, and a higher pass rate on the first attempt. It is interesting to see how different collaborative problem-based learning groups approached the same problem in distinct ways, resulting in varied outcomes. An interesting feedback from the participants was their curiosity about the origin and definition of the problem parameters. This led to the incorporation of a component where participants construct their own problem through a combination of independent investigation and collaborative problem-based learning. Furthermore, module has had a significant positive impact on participants' learning, allowing them to enhance their understanding by comparing their solutions with industrial solutions. The feedback from the participants of this module, particularly from

the industry, has played a significant role in refining the module, enhancing industrial tools, and advancing the state of the art in modeling and timing analysis of embedded software architectures.

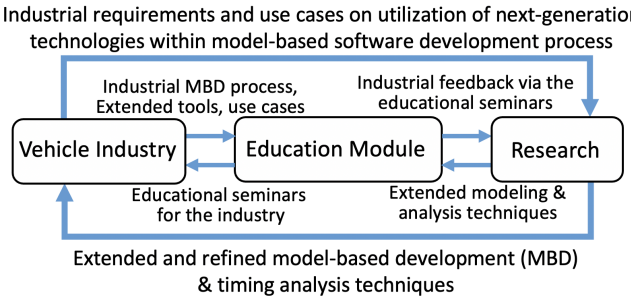

Figure 20: Collaboration resulting in the refinement of the software development approach in the educational module.

Acknowledgments

This work is supported by the Swedish Governmental Agency for Innovation Systems (VINNOVA) via the INTERCONNECT, PROVIDENT & FLEXATION projects, and by the Swedish Knowledge Foundation (KKS) via the SEINE project. The authors thank several persons in the context of this work: Prof. Reinder J. Bril for facilitating the module at the Technical University of Eindhoven, The Netherlands; Prof. Henri Muccini and Prof. Alfonso Pierantonio for facilitating the module at the University of L'Aquila, Italy; Prof. Goran Martinovic and Prof. Zdravko Krpic for facilitating the module at J. J. Strossmayer University of Osijek, Croatia; Prof. Matthias Becker (KTH Royal Institute of Technology, Sweden), Prof. Mikael Sjödin, and Prof. Alessio Bucaioni (Mälardalen University, Sweden) for providing valuable input and feedback. The author also thanks industrial partners, in particular, Volvo Construction Equipment, HIAB, BAE Systems, and Arcticus Systems, for providing their valuable input throughout the years.

References

[1] [n. d.]. Time-Sensitive Networking (TSN) Task Group, TSN standards, https://1.ieee802.org/tsn.
[2] [n. d.]. Henk G. Schmidt. A Brief History of Problem-based Learning. In: O'Grady, G., Yew, E., Goh, K., Schmidt, H. (eds) One-Day, One-Problem. Springer, Singapore, 2012, https://doi.org/10.1007/978-981-4021-75-3.
[3] [n. d.]. EAST-ADL Specification, http://www.east-adl.info/Specification/V2.1.12/EAST-ADL-Specification_V2.1.12.pdf.
[4] [n. d.]. Fraunhofer ESK, Safe Adaptive Software for Fully Electric Vehicles, https://www.iks.fraunhofer.de/en/projects/safeadapt.html.
[5] [n. d.]. *Land Rover Newsroom.* https://media.jaguarlandrover.com/news/2019/04/jaguar-land-rover-finds-teenagers-writing-code-self-driving-future.
[6] S. Ain. 2024. Safeguarding the Code That Drives Modern Vehicles. *Cyber Defense Magazine* (2024). https://www.cyberdefensemagazine.com/safeguarding-the-code-that-drives-modern-vehicles.
[7] Mohammad Ashjaei, Lucia Lo Bello, Masoud Daneshtalab, Gaetano Patti, Sergio Saponara, and Saad Mubeen. 2021. Time-Sensitive Networking in automotive embedded systems: State of the art and research opportunities. *Journal of Systems Architecture* 117 (2021), 102–137.
[8] Len Bass, Paul Clements, and Rick Kazman. 2012. *Software Architecture in Practice* (3rd ed.). Addison-Wesley Professional.
[9] Matthias Becker, Dakshina Dasari, Saad Mubeen, Moris Behnam, and Thomas Nolte. 2017. End-to-end timing analysis of cause-effect chains in automotive embedded systems. *Journal of Systems Architecture* 80 (2017), 104 – 113.
[10] Matthias Becker, Dakshina Dasari, Saad Mubeen, Moris Behnam, and Thomas Nolte. 2018. Analyzing end-to-end delays in automotive systems at various levels of timing information. *ACM SIGBED Review* 14, 4 (Jan. 2018), 8–13.
[11] Alessio Bucaioni, Saad Mubeen, Jhon Lundbäck, Kurt-Lennart Lundbäck, Jukka Mäki-Turja, and Mikael Sjödin. 2014. From Modeling to Deployment of Component-Based Vehicular Distributed Real-Time Systems. In *11th International Conference on Information Technology: New Generations.* 649–654.
[12] N. Feiertag, K. Richter, J. Nordlander, and J. Jonsson. 2008. A Compositional Framework for End-to-End Path Delay Calculation of Automotive Systems under Different Path Semantics. In *CRTS Workshop.*
[13] S. Fürst and M. Bechter. 2016. AUTOSAR for Connected and Autonomous Vehicles: The AUTOSAR Adaptive Platform. In *46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W).*
[14] K. Hänninen et al. 2008. The Rubus Component Model for Resource Constrained Real-Time Systems. In *IEEE Symposium on Industrial Embedded Systems.*
[15] Bahar Houtan, Mehmet Onur Aybek, Mohammad Ashjaei, Masoud Daneshtalab, Mikael Sjödin, and Saad Mubeen. 2022. End-to-end Timing Model Extraction from TSN-Aware Distributed Vehicle Software. In *Euromicro Conference Series on Software Engineering and Advanced Applications.*
[16] International Organization for Standardization (ISO). 2011. ISO 26262: Road Vehicles - Functional Safety.
[17] ISO 11898-1. 1993. Road Vehicles interchange of digital information controller area network (CAN) for high-speed communication, ISO Standard-11898. (1993).
[18] Xu Ke, K. Sierszecki, and C. Angelov. 2007. COMDES-II: A Component-Based Framework for Generative Development of Distributed Real-Time Control Systems. In *13th International Conference on Embedded and Real-Time Computing Systems and Applications.*
[19] Ramin Tavakoli Kolagari, DeJiu Chen, Agnes Lanusse, Renato Librino, Henrik Lönn, Nidhal Mahmud, Chokri Mraidha, Mark-Oliver Reiser, Sandra Torchiaro, Sara Tucci-Piergiovanni, Tobias Wägemann, and Nataliya Yakymets. 2015. Model-Based Analysis and Engineering of Automotive Architectures with EAST-ADL: Revisited. *Int. J. Concept. Struct. Smart Appl.* 3, 2 (2015), 25–70.
[20] L. Lo Bello, R. Mariani, S. Mubeen, and S. Saponara. 2019. Recent Advances and Trends in On-Board Embedded and Networked Automotive Systems. *IEEE Transactions on Industrial Informatics* 15, 2 (2019).
[21] Saad Mubeen, Mohammad Ashjaei, and Mikael Sjödin. 2019. Holistic Modeling of Time Sensitive Networking in Component-Based Vehicular Embedded Systems. In *45th Euromicro Conference on Software Engineering and Advanced Applications.*
[22] S. Mubeen, H. Lawson, J. Lundbäck, M. Gålnander, and K. L. Lundbäck. 2017. Provisioning of Predictable Embedded Software in the Vehicle Industry: The Rubus Approach. In *IEEE/ACM 4th International Workshop on Software Engineering Research and Industrial Practice (SER&IP).*
[23] S. Mubeen, J. Mäki-Turja, and M. Sjödin. 2014. Communications-Oriented Development of Component-Based Vehicular Distributed Real-Time Embedded Systems. *Journal of Systems Architecture* 60, 2 (2014), 207–220.
[24] Chrissi Nerantzi. 2014. A personal journey of discoveries through a DIY open course development for professional development of teachers in Higher Education. *Journal of Pedagogic Development* 4, 2 (2014), 42–58.
[25] K. Petersen, D. Badampudi, S. M. A. Shah, K. Wnuk, T. Gorschek, E. Papatheocharous, J. Axelsson, S. Sentilles, I. Crnkovic, and A. Cicchetti. 2018. Choosing Component Origins for Software Intensive Systems: In-House, COTS, OSS or Outsourcing?—A Case Survey. *IEEE Transactions on Software Engineering* 44, 3 (March 2018), 237–261.
[26] K. Richter, D. Ziegenbein, M. Jersak, and R. Ernst. 2002. Model Composition for Scheduling Analysis in Platform Design. In *Proceedings of the 39th Annual Design Automation Conference* (New Orleans, Louisiana, USA) *(DAC '02).* 287–292.
[27] S. Mubeen, J. Mäki-Turja, M. Sjödin. 2013. 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* 10, 1 (2013).
[28] S. Mubeen, T. Nolte, M. Sjödin, J. Lundbäck, and K. Lundbäck. 2019. Supporting timing analysis of vehicular embedded systems through the refinement of timing constraints. *Software & Systems Modeling* 18, 1 (2019).
[29] J. Schroeder, C. Berger, A. Knauss, H. Preenja, M. Ali, M. Staron, and T. Herpel. 2017. Predicting and Evaluating Software Model Growth in the Automotive Industry. In *IEEE International Conference on Software Maintenance and Evolution.*
[30] S. Sentilles, A. Vulgarakis, T. Bures, J. Carlson, and I. Crnkovic. [n. d.]. A Component Model for Control-Intensive Distributed Embedded Systems. In *11th International Symposium on Component Based Software Engineering, 2008.*
[31] L. Sha, T. Abdelzaher, K.-E. Årzén, A. Cervin, T. P. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. P. Lehoczky, and A. K. Mok. 2004. Real Time Scheduling Theory: A Historical Perspective. *Real-Time Systems* 28, 2/3 (2004), 101–155.
[32] The AUTOSAR Consortium. 2016. AUTOSAR Technical Overview. In *Version 4.3.* http://autosar.org.
[33] Mark Utting, Bruno Legeard, Fabrice Bouquet, Elizabeta Fourneret, Fabien Peureux, and Alexandre Vernotte. 2016. Chapter Two - Recent Advances in Model-Based Testing. Advances in Computers, Vol. 101. 53–120.
[34] Tassio Vale, Ivica Crnkovic, Eduardo Santana de Almeida, Paulo Anselmo da Mota Silveira Neto, YguaratÃ£ Cerqueira Cavalcanti, and Silvio Romero de Lemos Meira. 2016. Twenty-eight years of component-based software engineering. *Journal of Systems and Software* 111 (2016), 128 – 148.