On Model-Based Development of Embedded Software for Evolving Automotive E/E Architectures

Alessio Bucaioni, John Lundbäck, Mikael Sjödin, and Saad Mubeen

Abstract

Fueled by an increasing demand for computational power and high data-rate low-latency on-board communication, the automotive electrical and electronic architectures are evolving from distributed to consolidated domain and centralised architectures. Future electrical and electronic automotive architectures are envisioned to leverage heterogeneous computing platforms, where several different processing units will be embedded within electronic control units. These powerful control units are expected to be connected by high-bandwidth and low-latency on-board backbone networks. This paper draws on the industrial collaboration with the Swedish automotive industry for tackling the challenges associated to the model-based development of predictable embedded software for contemporary and evolving automotive E/E architectures.

Keywords

Automotive software · Electrical and electronic automotive architectures · Model-based development methodologies · Embedded systems

95.1 Introduction

In the past decades, automotive software has been evolving at a staggering pace [22]. Advanced Driver Assistance Systems (ADAS) and other advanced features in contemporary and upcoming automotive software require high levels of computational power and high data-rate low-latency on-board communication that is well beyond the capacity of traditional single-core Electronic Control Units (ECUs) and on-board buses/networks respectively. One important consequence of this trend is that traditional distributed Electrical/Electronic (E/E) architectures are giving way to consolidated domain and centralised E/E architectures [1, 2]. The progression and evolution of the automotive E/E architectures is depicted in Fig. 95.1. While consolidated domain E/E architectures are realised by employing multi-core processors, centralised automotive E/E architectures are envisioned to leverage heterogeneous computing platforms, e.g., containing Central Processing Units (CPUs), Graphical Processing Units (GPUs) and Field Programmable Gate Arrays (FPGAs), which are connected by high-bandwidth and low-latency on-board backbone networks such as Time Sensitive Networking (TSN) [22].

The introduction of heterogeneous computing platforms has opened up several challenges including modelling of heterogeneous hardware architectures, modelling of software architecture, software to hardware allocation and ensuring quality, scheduling, timing and performance analysis of the software architectures, just to mention a few [8]. For instance, the support for modelling heterogeneous hardware architectures is a challenging task mainly because of the requirement of data and memory management in a predictable way as well as the requirement of satisfying real-time constraints at the design time [8]. The state-of-the-art model-based software development methodologies for automotive embedded systems are unable to address all these challenges, as depicted in Fig. 95.1. One crucial step for shifting the current model-based software development methodologies into the new of domain and centralised E/E architectures, is to provide modelling languages with support for describing the software architecture, the heterogeneous execution platforms

A. Bucaioni (✉) · M. Sjödin · S. Mubeen
Mälardalen University, Västerås, Sweden
e-mail: alessio.bucaioni@mdh.se; mikael.sjodin@mdh.se; saad.mubeen@mdh.se

J. Lundbäck
Arcticus Systems, Järfälla, Sweden
e-mail: john.lundback@arcticus-systems.com

S. Latifi (ed.), *17th International Conference on Information Technology–New Generations (ITNG 2020)*, Advances in Intelligent Systems and Computing 1134,
https://doi.org/10.1007/978-3-030-43020-7_95

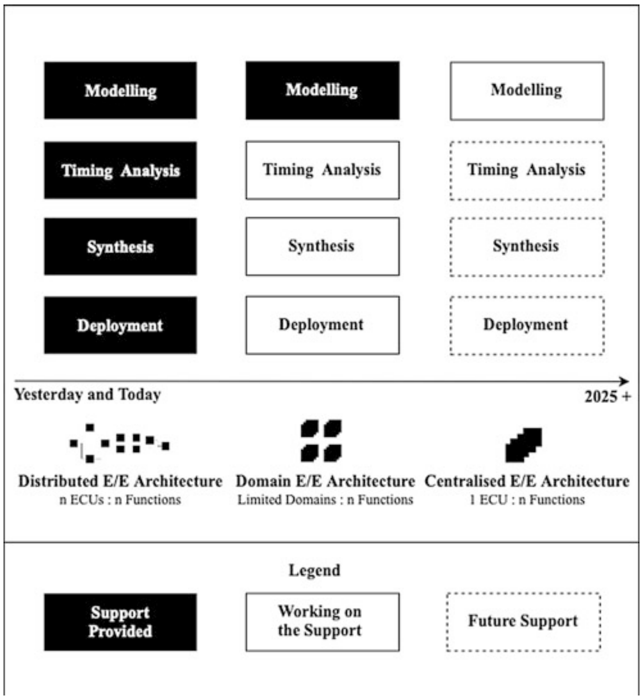


Fig. 95.1 Evolution of automotive E/E architectures and supporting model-based development methodologies

and the software to hardware allocation. Note that many automotive software functions are time critical, i.e., they are required to provide logically correct responses at right times that conform to the specified timing constraints. Hence, the modelling for future automotive E/E architectures should be supported by timing analysis engines [10, 27, 29] and predictable run-time environments [3, 8, 24].

This paper draws on the industrial collaboration with the Swedish automotive industry for reporting on the advancement of model-based methodologies for the development of automotive software with respect to the evolution of automotive E/E architectures. In particular, this paper discusses ongoing works with respect to architecture design (hardware and software co-design), support for timing analysis of the software architectures and provision of predictable run-time environment for distributed and domain E/E architectures. What is more, this paper describes a pragmatic vision on how to tackle the challenges of hardware and software co-design for centralised E/E architectures.

The rest of the paper is organised as follows. Section 95.2 describes a comparison between existing related approaches documented in the literature and our solution. Section 95.3 presents an industrial approach to support model-based software development on distributed automotive E/E architectures. Moreover, it discusses the partially available support for model-based software development on domain automotive E/E architectures. This section also discusses future plans for the software development on centralised automotive E/E architectures. Finally Sect. 95.4 concludes the paper and discusses the future work.

95.2 Related Work

In recent years, automotive software has been on the forefront of many software engineering advances including model-based development methodologies and real-time techniques [22]. This section presents the related research on modelling languages and approaches that are supported by end-to-end timing analysis and *timing predictable* run-time environments for distributed and domain automotive E/E architectures. Note that timing predictability is a well-known term in time-critical systems domain, where it is defined as a system-level property. A system is considered timing predictable under a set of assumptions if it is possible to demonstrate or prove at the design time that all timing requirements specified on the system are satisfied and that the system will certainly meet its timing when executed [17, 20, 25, 33, 34].

EAST-ADL is an architecture description language for the specification of automotive E/E architectures. It uses a multi-layer approach, where each layer describes the automotive architecture at a different abstraction level and from a different perspective [7]. At higher layers, EAST-ADL does not allow to explicitly model execution platforms being multi-core or heterogeneous. This means, that EAST-ADL supports modelling of software architectures for distributed automotive E/E architectures but for domain and centralised automotive E/E architectures. There are several works that allow timing analysis of software architectures that are modelled with EAST-ADL for distributed automotive E/E architectures [12, 13, 29]. However, there is no support for EAST-ADL to perform timing analysis to verify timing predictability of the software architectures on domain and centralised automotive E/E architectures. The EAST-ADL development methodology proposes to use domain-specific modelling languages (DSL) at the lower levels of abstraction, notably AUTOSAR [4], Rubus Component Model (RCM) [18, 24], and so forth. As a consequence, the work described in this paper can be considered as complementary to EAST-ADL.

AUTOSAR was created as an industrial initiative to provide a standardised software architecture at the implementation layer of the EAST-ADL methodology. AUTOSAR distinguishes among three software layers namely Application, Runtime Environment and Basic Software. In its last definition, AUTOSAR provides limited support for automotive software on multi-core platforms while it provides no-support for automotive software on heterogeneous platforms. This means that AUTOSAR supports modelling of software architectures and run-time environments for distributed and domain automotive E/E architectures but not for centralised automotive E/E architectures.

Based on AUTOSAR, APP4MC [5] is an open source platform for engineering embedded multi- and many-core software systems. This platform is mainly used within the automotive domain and relies on the AMALTHEA datamodels which provide support for modelling, e.g., hardware, software, mapping, stimuli, events, among others. AMALTHEA datamodels allow to describe both multi-core and heterogeneous platforms. Compared to the modelling approach leveraged in this paper, the AMALTHEA approach employs an extensive and explicit modelling of the execution platform (in terms of processing units, caches, memories, etc.) and of all the components of the automotive system including, e.g., operating system.

Similar to APP4MC, Distributed Real-time Architecture for Mixed criticality Systems (DREAMS) [6] is an European project which aims at developing a cross-domain architecture and design tools for networked complex systems supporting application subsystems with different criticality. DREAMS delivers metamodels implementing different views including, e.g., *logical*, *physical* and *temporal*.

Several works are based on the use of general-purpose languages such as UML as alternatives to automotive-specific languages. CHESS and GASPARD are examples of UML-based languages. The former allows to model complex component-based embedded systems in terms of their platform(s) and relevant properties, such as timing [15]. GASPARD is mostly used for the design of parallel embedded systems [16].

AADL [30] is an architecture description language conceived for the avionics domain, but it has been increasingly used for modelling embedded systems in general.

In crux, the state-of-the-art modelling approaches, timing analysis techniques and run-time environments provide a good, limited and no support for distributed, domain and centralised automotive E/E architectures respectively.

95.3 Model-Based Development of Automotive Software

In this section, we discuss the advancement of model-based methodologies for the development of automotive software with respect to the evolution of automotive E/E architectures shown in Fig. 95.1. In doing so, we draw on the industrial collaboration with Arcticus Systems,[1] a Swedish tool provider for international companies in the automotive industry such as Volvo Construction Equipment,[2] BAE Systems,[3] just to mention a few. During the last decades, the research collaboration between Arcticus Systems and Mälardalen University led to the definition of a model-based development methodology which is embodied in the Rubus Integrated Component model development Environment (Rubus-ICE). Rubus-ICE is based around the Rubus Component Model (RCM) which is a modelling language for distributed real-time embedded systems [14]. The methodology embodied in Rubus-ICE consists of four major phases: modelling, timing analysis, synthesis and deployment as shown in Fig. 95.1. As all the phases are carried out within Rubus-ICE, this methodology avoids explicit interoperability management and reduces time and cost overheads.

95.3.1 Model-Based Development of Automotive Software on Distributed Automotive E/E Architectures

Rubus-ICE provides a full-fledged model-based methodology for the development of automotive software on distributed E/E architectures. The development support includes modelling of the automotive software architectures and timing information (timing properties, requirements and constraints), end-to-end timing analysis of the software architectures, automatic generation of timing verified code from the software architectures, deployment and execution on predictable run-time environment. Figure 95.2 shows a screenshot of an example of a real-time system modelled and analysed on a distributed automotive E/E architecture.

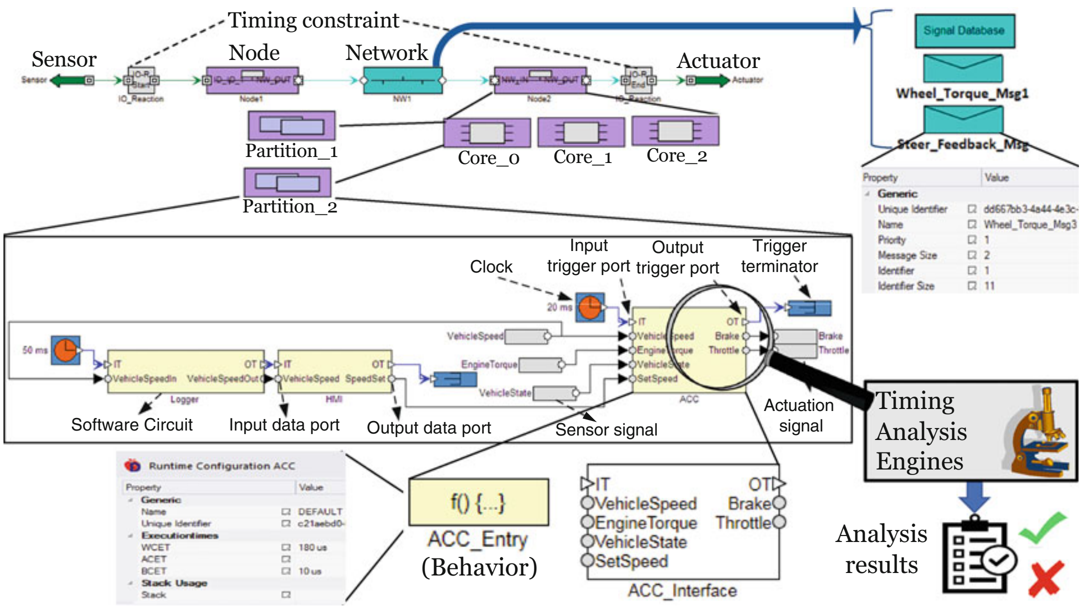
95.3.1.1 Modelling

Rubus-ICE fully supports modelling of software architectures on distributed automotive E/E architectures. Within Rubus-ICE, the automotive software architecture and its timing properties are modelled with RCM. In RCM, a Software Circuit (SWC) is the lowest-level hierarchical element and it represents the basic component that encapsulates one or more software functions. For example, the yellow boxes in Fig. 95.2, namely Logger, HMI and ACC represent three SWCs. Two or more SWCs may be encapsulated into a software assembly (ASM) for constructing the system at different hierarchical levels. An SWC has the run-to-completion semantics. In RCM, the interaction between SWCs is expressed in terms of data and control flow, separately. The SWCs communicate with each other via data ports. The component model facilitates analysis and reuse of components in

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

[2] https://www.volvoce.com/.

[3] https://www.baesystems.com/en/our-companies/our-businesses/platforms-and-services/locations/sweden.



Property	Value
Generic	
Unique Identifier	dd667bb3-4a44-4e3c-
Name	Wheel_Torque_Msg3
Priority	1
Message Size	2
Identifier	1
Identifier Size	11

Runtime Configuration ACC

Property	Value
Generic	
Name	DEFAULT
Unique Identifier	c21aebd0-
Executiontimes	
WCET	180 us
ACET	
BCET	10 us
Stack Usage	
Stack	

Fig. 95.2 Example of software development for real-time embedded systems on various Automotive E/E architectures using Rubus-ICE

different contexts by separating functional code from the infrastructure that implements the execution environment. RCM allows modelling of single-core processing units by means of node elements as shown by the model of two Node1 in Fig. 95.2. The nodes can be distributed, in which case they are connected by one or more models of networks. This is the case of the node elements Node1 and Node2 in Fig. 95.2 which are connected by the network element NW1. RCM supports modelling of various types of in-vehicle networks, including broadcast networks like Controller Area Network (CAN) [19] and its higher level protocols [28] and point-to-point networks like Ethernet Audio Video Bridging (AVB) [9] and TSN [26].

95.3.1.2 Timing Analysis

RCM allows expressing real-time requirements and properties on the software architecture of distributed automotive E/E architectures. To this end, the designer has to express real-time properties of SWCs, such as worst-case execution times (WCETs), stack usage, etc. The WCETs of the SWCs can be determined by using static WCET analysis tools such as SWEET [21]. The plugin framework in Rubus-ICE supports integration of such tools. The SWCs that are activated by periodic triggering sources, e.g., the Logger and ACC components in Fig. 95.2, are statically scheduled using the Rubus off-line scheduler. The scheduler constructs a schedule taking into account the specified real-time constraints. For event-triggered SWCs, response-time analysis [32] is performed and the calculated response times are compared with the specified timing requirements. The supported analysis, among others, includes distributed end-to-end response-time and delay analyses [27] and shared stack analysis [11].

95.3.1.3 Synthesis

While the software development using Rubus-ICE is independent of the underlying operating system (OS), code synthesizers are not. For this reason, Rubus-ICE accompanies the Rubus Real-Time OS (RTOS) designed for predictable execution of the software architecture. The Rubus RTOS supports both time- and event-triggered execution of tasks.[4] It optimises the run-time architecture by using the hybrid scheduling combining the static cyclic scheduling and the fixed-priority preemptive scheduling [23]. The Rubus RTOS has been ported to several different commercial-off-the-shelf processors [24].

95.3.1.4 Deployment

Within Rubus-ICE, the deployment phase involves both software and hardware platforms. The Rubus RTOS provides for the software platform. Although the software platform is OS dependent, it should be noted that the software architecture and corresponding synthesised code can be easily adapted and deployed to any RTOS. Similarly, any processing unit capable of running an RTOS, e.g., IBM's PowerPC or ARM processor can serve as the hardware platform for deployment.

95.3.2 Model-Based Development of Automotive Software on Domain Automotive E/E Architectures

Domain E/E architectures employ more powerful multi- and many-core processing units for replacing the constellation

[4]Tasks are run-time entities, whereas SWCs are equivalent design-time entities.

of single-core units realising distributed E/E architectures. Currently, Rubus-ICE allows modelling of software architectures and specification of timing information of automotive software systems that are deployed on these architectures. For example, Fig. 95.2 shows an RCM model of a software architecture of a real-time system that is deployed on a tri-core node with multiple partitions per core (Node2). The support for timing analysis and predictable run-time support for real-time systems on these architectures is an ongoing work.

95.3.2.1 Modelling

RCM fully supports modelling of software architectures on domain automotive E/E architectures. In fact, in our previous work, we extended RCM for modelling automotive software on multi-core processing units [14]. The extension included the introduction of new modelling elements for describing multi-core processing units, software applications and their criticality levels, and the software to hardware allocation. Figure 95.3 shows the RCM extensions supporting multi-core platforms. Multi-core processing units are modelled in terms of node, core and partition elements. For instance, the node element Node2 in Fig. 95.2 is modelled as a tri-core processor comprising the core elements Core_0, Core_1 and Core_2. In turns, the core element Core_0 has two partition elements, namely Partition_1 and Partition_2. Within RCM, partition elements isolate parts of software from each others in both *time and space*. Isolation in time means that each partition gets a reserved share of the core processing time while isolation in space means that the memory available to each core is divided among its partitions. The extended RCM leverages an allocation mechanism which replaces the use of structural containment relations in favour of more flexible relationships among software and hardware elements. In particular, such relationships can only be specified among node, core, and partition elements and application, mode, assembly, and SWC elements.

95.3.2.2 Timing Analysis

Today Rubus-ICE uses offline scheduling to schedule software architectures on multi-core platforms. Moreover, resource partitioning techniques are considered for managing the shared resources such as cache memories and system bus. Hence, the scheduled software architecture is correct by construction from timing perspective.

95.3.2.3 Synthesis

Generation of code from the software architectures of the applications that are deployed on domain automotive E/E architectures is an ongoing work.

95.3.2.4 Deployment

The isolation in time and space is supported by the run-time layer, where the Rubus multi-core hypervisor uses resource-isolation techniques for arbitration of intra- and inter-core shared resources. As a result of using isolation techniques, each core and partition becomes (virtually) independent meaning that they can be seen as a single-core processor equivalent with dedicated system resources. One notable advantage of this is that the overall system becomes simpler to model as there is no need to explicitly model memories, I/Os and other shared resources in the software architecture.

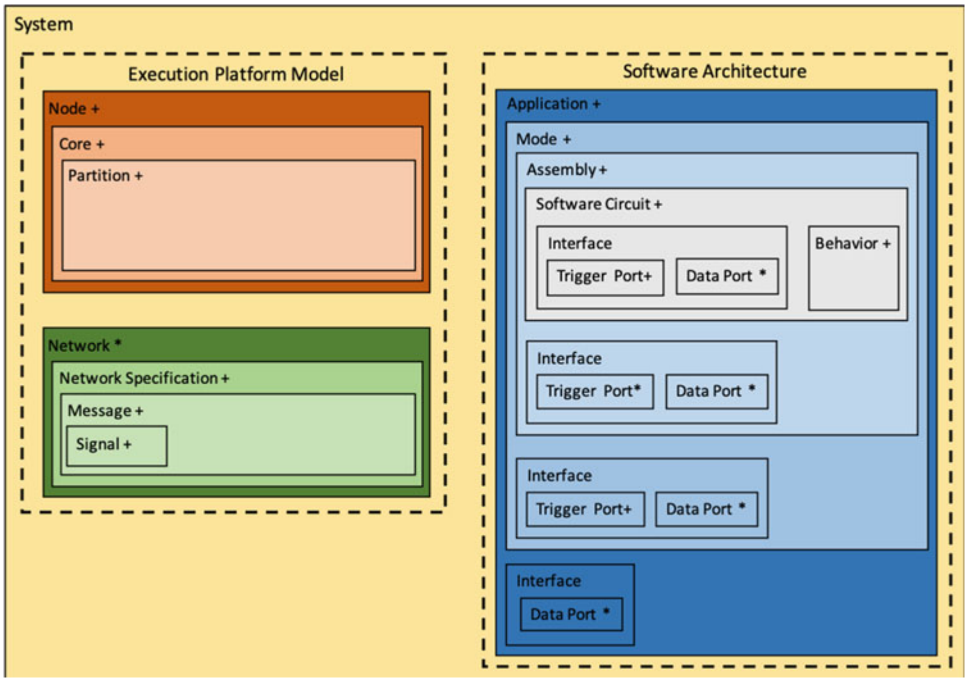


Fig. 95.3 RCM extensions supporting multi-core platforms

95.3.3 Model-Based Development of Automotive Software on Centralised Automotive E/E Architectures

Centralised E/E architectures are envisioned to leverage heterogeneous hardware comprising of certified traditional processors and general-purpose high-performance processors with accelerators. As a result, centralised E/E architecture will require the integration of heterogeneous software with respect to, e.g., workloads, activation semantics, data-flow semantics, real-time requirements and safety requirements [31]. What is more, they open up to several development challenges including software architecture and quality, scheduling and hardware [8].

Currently, RCM supports the specification of heterogeneous software with respect to real-time properties and requirements, safety requirements and criticality levels (different Automotive Safety Integrity Levels (ASILs) A to D according to the ISO 26262 functional safety standard for road vehicles), activation semantics (time triggered, event triggered), data-flow semantics (synchronous, independent activation, task chains) and workloads. An on-going work is the extension of RCM with fine-grained modelling elements for the specification of heterogeneous hardware. This extension will include elements such as GPU, FPGA, memory, cache, etc. Support for timing analysis, synthesis and deployment are future works. To the best of our knowledge, there is no modelling framework or methodology that supports all the above mentioned development steps (shown in Fig. 95.2) for centralised automotive E/E architectures.

95.4 Conclusion and Future Work

This paper presents a component model, a development methodology and an integrated development environment that are used in the automotive industry. The tool chain, Rubus-ICE, supports model- and component-based development of embedded software FPR evolving automotive Electrical/Electronic (E/E) architectures. The paper also demonstrated how current industrial model-based methodologies support modelling, timing-analysis, synthesis and deployment of embedded software on distributed automotive E/E architectures. The paper also discussed on-going works for enriching industrial model-based methodologies with support for architecture design (hardware and software co-design), support for end-to-end timing analysis of the software architectures and provision of predictable run-time environment for embedded software on automotive domain and centralised E/E architectures. We identify that a full-fledged model-based software development methodology, development environment and run-time support for predictable automotive software on domain and centralised automotive E/E architectures is still missing. One line of future work encompasses provisioning of timing analysis, synthesis and deployment of embedded software on domain automotive E/E architectures. Another line of future work includes the definition of a reference architecture for heterogeneous platforms employed in the realisation of centralised automotive E/E architectures.

Acknowledgments The work in this paper is supported by the Swedish Knowledge Foundation (KKS) via the projects A-CPS, HERO and DPAC, and by the Swedish Governmental Agency for Innovation Systems (VINNOVA) via the projects PANORAMA and DESTINE. The authors would like to thank the industrial partners, especially Arcticus Systems and Volvo, Sweden.

References

1. Berger, R.: Consolidation in vehicle electronic architectures. In: Think: AACT (2015). https://www.rolandberger.com/en/Publications/pub_consolidation_in_vehicle_electronic_architectures.html
2. Zinner, H., Brand, J., Hopf, D.: Automotive E/E Architecture evolution and the impact on the network. In: IEEE802 Plenary, 802.1 TSN, Continental AG (2019). Available http://ieee802.org/1/files/public/docs2019/dg-zinner-automotive-architecture-evolution-0319-v02.pdf
3. The AUTOSAR consortium, AUTOSAR requirements on runtime environment, AUTOSAR CP Release 4.3.1 (2017). https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_SRS_RTE.pdf
4. The AUTOSAR consortium, AUTOSAR technical overview, Version 4.3 (2016). http://autosar.org
5. AMALTHEA Project Profile (2017). http://www.amalthea-project.org
6. DREAMS—distributed REal-time architecture for mixed criticality systems (2019). http://www.dreams-project.eu
7. East-ADL domain model specification, deliverable d4.1.1 (2010). http://www.atesst.org/home/liblocal/docs/ATESST2_D4.1.1_EAST-ADL2-Specification_2010-06-02.pdf
8. Andrade, H., Schroeder, J., Crnkovic, I.: Software deployment on heterogeneous platforms: a systematic mapping study. IEEE Trans. on Softw. Eng., 1–1 (2019)
9. Ashjaei, M., Mubeen, S., Lundbäck, J., Gålnander, M., Lundbäck, K., Nolte, T.: Modeling and timing analysis of vehicle functions distributed over switched ethernet. In: IECON 2017—43rd Annual Conference of the IEEE Industrial Electronics Society, pp. 8419–8424 (2017). https://doi.org/10.1109/IECON.2017.8217478
10. Becker, M., Dasari, D., Mubeen, S., Behnam, M., Nolte, T.: End-to-end timing analysis of cause-effect chains in automotive embedded systems. J. Syst. Archit. **80**, 104–113 (2017). https://doi.org/10.1016/j.sysarc.2017.09.004
11. Bohlin, M., Hänninen, K., Mäki-Turja, J., Carlson, J., Sjödin, M.: Bounding shared-stack usage in systems with offsets and precedences. In: 20th Euromicro Conference on Real-Time Systems (2008)
12. Bucaioni, A., Addazi, L., Cicchetti, A., Ciccozzi, F., Eramo, R., Mubeen, S., Sjödin, M.: Moves: a model-driven methodology for vehicular embedded systems. IEEE Access **6**, 6424–6445 (2018). https://doi.org/10.1109/ACCESS.2018.2789400
13. Bucaioni, A., Mubeen, S., Cicchetti, A., Sjödin, M.: Exploring timing model extractions at east-adl design-level using model trans-

formations. In: 2015 12th International Conference on Information Technology - New Generations, pp. 595–600 (2015). https://doi.org/10.1109/ITNG.2015.100
14. Bucaioni, A., Cicchetti, A., Ciccozzi, F., Mubeen, S., Sjödin, M.: Technology-preserving transition from single-core to multi-core in modelling vehicular systems. In: Springer (ed.) 13th European Conference on Modelling Foundations and Applications (2017). http://www.es.mdh.se/publications/4750-
15. Cicchetti, A., Ciccozzi, F., Mazzini, S., Puri, S., Panunzio, M., Zovi, A., Vardanega, T.: Chess: a model-driven engineering tool environment for aiding the development of complex industrial systems. In: Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering. pp. 362–365. ACM, New York (2012)
16. Gamatié, A., Le Beux, S., Piel, E., Ben Atitallah, R., Etien, A., Marquet, P., Dekeyser, J.L.: A model-driven design framework for massively parallel embedded systems. ACM Trans. Embed. Comput. Syst. **10**(4), 39:1–39:36 (2011)
17. Grund, D., Reineke, J., Wilhelm, R.: A template for predictability definitions with supporting evidence. In: Bringing Theory to Practice: Predictability and Performance in Embedded Systems. OpenAccess Series in Informatics, vol. 18, pp. 22–31. Dagstuhl, Germany (2011). https://doi.org/10.4230/OASIcs.PPES.2011.22
18. Hänninen, K., Mäki-Turja, J., Sjödin, M., Lindberg, M., Lundbäck, J., Lundbäck, K.L.: The Rubus component model for resource constrained real-time systems. In: 3rd IEEE International Symposium on Industrial Embedded Systems (2011)
19. ISO 11898-1: Road Vehicles Interchange of Digital Information Controller Area Network (CAN) for High-speed Communication (1993)
20. Kirner, R., Puschner, P.: Time-predictable computing. In: Software Technologies for Embedded and Ubiquitous Systems, pp. 23–34. Springer, Berlin (2010)
21. Lisper, B.: SWEET—a tool for WCET flow analysis. In: 6th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation. pp. 482–485. Springer, Berlin (2014)
22. Lo Bello, L., Mariani, R., Mubeen, S., Saponara, S.: Recent advances and trends in on-board embedded and networked automotive systems. IEEE Trans. Ind. Inf. **15**(2), (2019). https://doi.org/10.1109/TII.2018.2879544
23. Mäki-Turja, J., Hänninen, K., Nolin, M.: Efficient development of real-time systems using hybrid scheduling. In: 9th Real-Time in Sweden (RTiS'07), pp. 157–163 (2007)
24. Mubeen, S., Lawson, H., Lundbäck, J., Gålnander, M., Lundbäck, K.: Provisioning of predictable embedded software in the vehicle industry: the rubus approach. In: 4th IEEE/ACM International Workshop on Software Engineering Research and Industrial Practice (2017)
25. Mubeen, S., Lisova, E., Feljan, A.V.: A perspective on ensuring predictability in time-critical and secure cooperative cyber physical systems. In: 2019 IEEE International Conference on Industrial Technology (ICIT), pp. 1379–1384 (2019). https://doi.org/10.1109/ICIT.2019.8754962
26. Mubeen, S., Ashjaei, M., Sjödin, M.: Holistic modeling of time sensitive networking in component-based vehicular embedded systems. In: Euromicro Conference on Software Engineering and Advanced Applications (2019). http://www.es.mdh.se/publications/5515-
27. Mubeen, S., Mäki-Turja, J., Sjödin, M.: Support for end-to-end response-time and delay analysis in the industrial tool suite: issues, experiences and a case study. In: Computer science and information systems, vol. 10(1), pp. 453–482 (2013)
28. Mubeen, S., Mäki-Turja, J., Sjödin, M.: Integrating mixed transmission and practical limitations with the worst-case response-time analysis for controller area network. J. Syst. Softw. **99**, 66–84 (2015). https://doi.org/10.1016/j.jss.2014.09.005. http://www.sciencedirect.com/science/article/pii/S0164121214001952
29. Mubeen, S., Nolte, T., Sjödin, M., Lundbäck, J., Lundbäck, K.L.: Supporting timing analysis of vehicular embedded systems through the refinement of timing constraints. Softw. Syst. Model. **18**(1), 39–69 (2019). https://doi.org/10.1007/s10270-017-0579-8
30. Peter, H.F., David, P.G., Hudak, J.: The architecture analysis and design language (AADL): an introduction. Technical Report (2006)
31. Saidi, S., Steinhorst, S., Hamann, A., Ziegenbein, D., Wolf, M.: Future automotive systems design: research challenges and opportunities: special session. In: Proceedings of the IEEE International Conference on Hardware/Software Codesign and System Synthesis, p. 2 (2018)
32. Sha, L., Abdelzaher, T., Årzén, K.E., Cervin, A., Baker, T., Burns, A., Buttazzo, G., Caccamo, M., Lehoczky, J., Mok, A.K.: Real time scheduling theory: a historical perspective. Real-Time Syst. **28**(2-3), 101–155 (2004)
33. Stankovic, J.A., Ramamritham, K.: What is predictability for real-time systems? Real-Time Syst. **2**(4), 247–254 (1990)
34. Thiele, L., Wilhelm, R.: Design for timing predictability. Real-Time Syst. **28**(2), 157–177 (2004). https://doi.org/10.1023/B:TIME.0000045316.66276.6e