Towards a metamodel for the Rubus Component Model

Alessio Bucaioni, Antonio Cicchetti, and Mikael Sjödin

Mälardalen Real-Time Research Centre (MRTC)
School of Innovation, Design and Engineering (IDT)
Mälardalen University, Västerås, Sweden
{alessio.bucaioni,antonio.cicchetti,mikael.sjodin}@mdh.se

Abstract. *Component-Based Software Engineering has been recognized as an effective practice for dealing with the increasing complexity of the software for vehicular embedded systems. Despite the advantages it has introduced in terms of reasoning, design and reusability, the software development for vehicular embedded systems is still hampered by constellations of different processes, file formats and tools, which often require manual ad hoc translations. By exploiting the crossplay of Component-Based Software Engineering and Model-Driven Engineering, we take initial steps towards the definition of a seamless chain for the structural, functional and execution modeling of software for vehicular embedded systems. To this end, one of the entry requirements is the metamodels definition of all the technologies used along the software development. In this work, we define a metamodel for an industrial component model, Rubus Component Model, used for the software development of vehicular real-time embedded systems by several international companies. We focus on the definition of metamodeling elements representing the software architecture.*

Keywords: Component-Based Software Engineering, Model-Driven Engineering, Component-Based Software Systems, Vehicular Embedded Systems, Rubus Component Model

1 Introduction

During the last decades, industrial requirements on vehicular embedded systems have been constantly evolving causing an enlargement of the related software complexity: it has been estimated that current vehicles have more than 70 embedded systems running up to 100 million lines of code [14]. In this context, traditional software development processes have revealed strong limitations. On the one hand, industry needs efficient processes for reducing software development cost and time-to-market. On the other hand, most of the vehicular embedded systems present real-time properties, which have to be taken into account from the early stages of the development.

Component Based Software Engineering (CBSE) [15] has been acknowledged as an effective practice for dealing with the increasing software complexity. It promotes the development of the system at higher level of abstraction relying on the definition and reuse of atomic unit of composition, i.e., components. Also, CBSE allows to annotate components, at design time, with real-time properties

and constraints, e.g., worst-case execution time, enabling pre-run-time analysis, e.g., end-to-end response time and delay analysis [15].

Several component-based development processes have been introduced for improving the vehicular embedded systems software development. EAST-ADL, together with its follow-up initiatives, is the *de facto* standard for the software development of vehicular embedded systems. Among other contributions, EAST-ADL has standardized the terminology and promoted separation of concerns through a top-down development process, which makes use of four different abstraction layers. Despite the great initial reception, EAST-ADL is rarely adopted as it is. For instance, considering modern vehicle development, e.g., vehicles product line, it is very unlikely that vehicles are developed from scratch using top-down approaches. Contrariwise, they are mostly developed using a bottom-up strategy, reusing pre developed and tested components. In this context, the process defined by EAST-ADL finishes to hamper the software development, as the concepts used in each layer are designed for hiding non necessary information at higher and lower layers. While this can be effective within a top-down strategy - where the artifacts are enriched as they move forward towards the development layers - it is counterproductive when used within a bottom-up strategy - where low-level information, such as component's real time properties, need to be available at the earlier development stages. Also, the industrial vehicle software development is hindered by manual *ad hoc* translations, needed to integrate legacy systems and external tools: automation and tools integration have been acknowledged, by several projects [1], as key factors when dealing with extensive architectures as those for vehicular embedded systems. Information management, interoperability and traceability issues can not be fully solved using CBSE, as the discipline itself was not defined towards such aspects [15].

Model-Driven Engineering (MDE) is a discipline which promotes the separation of concerns by using different models for different concerns [16]. Unlike CBSE, MDE establishes precise relationships among models for the automatic generation of new models, change propagation and model-synchronization [16]. In this respect, MDE enhances software development targeting important development issues, such as information management, traceability, integration and interoperability [17].

We propose to exploit the crossplay of MDE and CBSE for realizing a seamless chain for the structural, functional and execution modeling of software for vehicular embedded systems. To this end, we believe one of the entry requirements is the metamodels definition of all the technologies used along the software development. In this work we define a metamodel for the Rubus Component Model (RCM), a component model (CM) used in the development of resource-constrained real-time vehicular embedded systems, focusing on the metamodeling elements representing the software architecture. As a proof of concept, we show a model transformation from RCM to AUTOSAR (RCM2AUTOSAR).

The rest of the paper is organized as follows. Section 2 presents the context of this work. Section 3 introduces the RCM metamodel. Section 4 shows the

[1] OSLC: http://open-services.net ; CRYSTAL: http://www.crystal-artemis.eu

RCM2AUTOSAR transformation while Section 5 discusses some related works. Finally, Section 6 draws conclusions and future works.

2 Context

In this section we present the context of this work by describing the four abstraction layers used in the software development of vehicular embedded systems. Additionally, we give some insights about RCM and the accompanying tool suite.

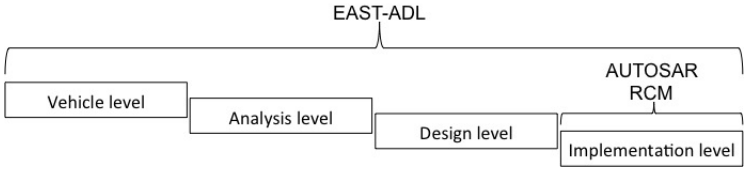


Fig. 1. The four abstraction layers as introduced by the EAST-ADL specification

2.1 Abstraction levels

EAST-ADL standardized a top-down development process composed of four different abstraction layers. Despite the top-down strategy is rarely used in industry, the abstraction layers and the related terminology have been fully adopted. The four abstraction levels are shown in Figure 1.

Vehicle level The vehicle level captures all the information regarding what the system is supposed to do. Feature models can be used for showing what the system provides and, eventually, how the product line is organized in terms of available assets. Feature models can be complemented with requirements. The vehicle layer is also known as End-to-End level as it serves to capture requirements and features on the end-to-end vehicle functionality.

Analysis level In the analysis level, vehicle functions are expressed using formal notations. The functionality are defined in terms of behaviors and interfaces. Yet, design and implementation details are omitted. At this stage, high level analysis for functional verification can be performed.

Design level In this level, the analysis-level artifacts are refined with more design-oriented details. While the analysis level does not differentiate among software, middleware abstraction and hardware architecture, the Design level explicitly separates this areas of the system implementation. Also, software functions to hardware allocation is expressed in this layer.

Implementation level In the implementation layer, the design-level artifacts are refined with implementation details. At this stage CMs, e.g., AUTOSAR, RCM, can be used to model the systems in terms of components and interactions among them. The output of this layer is a complete software architecture used for the code synthesis.

2.2 The Rubus Concept

Rubus [10] is a collection of methods, theories and tools for model- and component-based development of resource-constrained embedded real-time systems; it is developed by Arcticus Systems in collaboration with Mälardalen University. It is mainly used for the development of control functionality in vehicles by several international companies. The Rubus concept is based around the RCM[11] and its development environment Rubus-ICE [10]. Rubus-ICE includes:

- The Rubus Analysis Framework, for expressing real-time requirements and properties while modeling the system architecture;
- The Rubus Code Generator and Run-Time System, for synthesizing the code from the specified architecture;
- The Rubus SIMulation Model (RSIM), for simulating and testing the architecture at all the different hierarchical levels, e.g., components, Electronic Control Units (ECUs), subsystems, complete distributed system;
- The Rubus Execution Platform, for optimizing the run-time architecture.

With respect to the aforesaid four layers architecture, RCM is currently used in the implementation level as alternative/complement to AUTOSAR .

3 Providing a Metamodel to RCM

In this section, we present the RCM metamodel. [2] focusing on the metamodel definition of the architectural elements. For reading sake, we present the metamodel in three sections: Section 3.1 introduces the metamodel backbone, Section 3.2 introduces the metamodel elements for the data flow while Section 3.3 introduces the metamodel elements for the control flow.

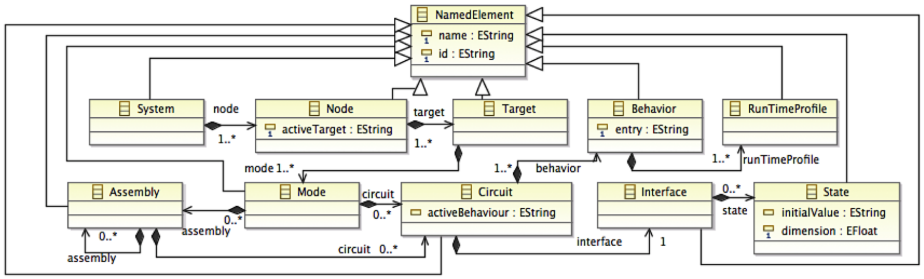


Fig. 2. Metamodel fragment for the backbone architectural elements

3.1 Backbone

Figure 2 shows the metamodel backbone. The top element is *System*, which acts as a container for the whole architecture. System, as all the elments in the metamodel, inherits from the abstract element *NamedElement*. A System element contains one or more *Node*(s). A Node is a hardware and operating-system independent abstraction of a *Target* and groups the software architecture elements which realize a certain function. Its attribute *activeTarget* defines which

[2] In this work, we do not seek to explain RCM; the interested reader may refer to [11]

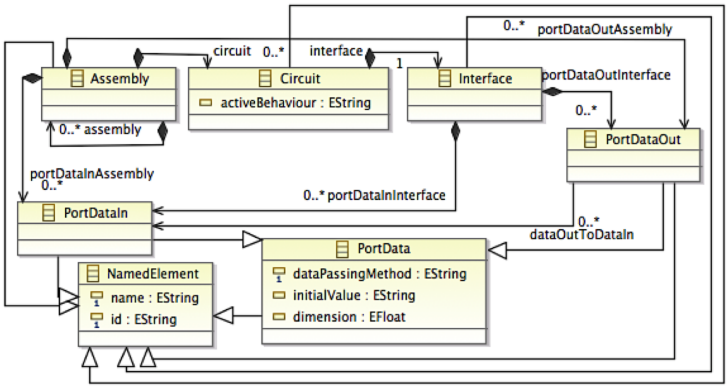
Target, among those specified, is active for a certain Node. A Target is a hardware and operating-system specific instance of a Node, which models the deployment of the software architecture, that is, it contains all the functions which have to be deployed on the same ECU. A Node can be realized by different Targets, depending from which hardware and operating system are considered for the deployment, i.e., PowerPC with Rubus Operating System, Simulated target with Windows operating system. A Target might contain one or more *Mode*(s). A Mode represents a specific application of the software, i.e., start-up mode, low power mode. A Mode might contains *Circuit*(s) and *Assembly*(ies). A Circuit is the lowest-level hierarchical element which encapsulates basic functions. It is composed by an *Interface*, which collects its data and triggering ports (Section 3.2 and Section 3.3), and one, or more, *Behavior*(s). A Circuit has the run-to-completion semantic, which means that, upon triggering, it reads data from the input ports, executes its behavior and writes data on the output ports. Its attribute *activeBehavior* specifies which Behavior, among those defined, is active. A Behavior represents the code to be executed. It has one attribute, *entry*, and it is composed by one or more *RunTimeProfile*(s), which define the Behavior execution and run-time properties for a specific platform. Interface might be composed by several *State*(s). A State is used for preserving data among the different executions of a behavior. A State has two attributes: *initialValue*, and *dimension*. An Assembly is used for grouping different Circuits or Assemblies; it does not add any semantic.

3.2 Data elements

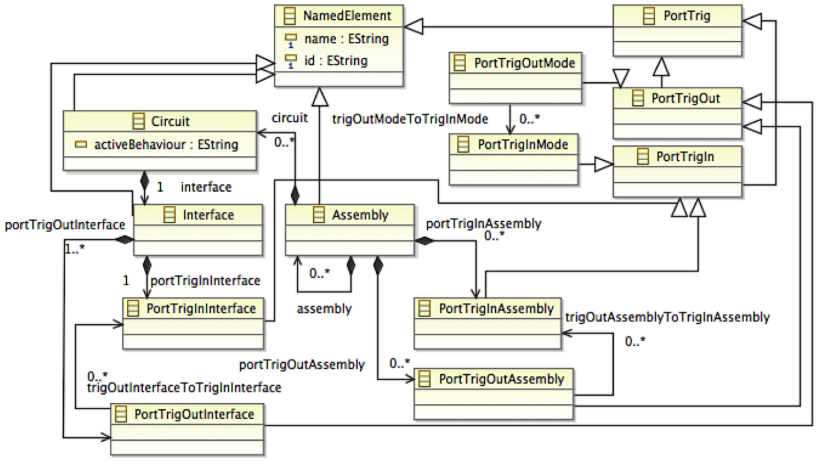
As aforesaid, RCM explicitly separates the data and the control flow. Figure 3(a) shows a metamodel fragment containing the architectural elements for modeling the data flow. *PortData* models a generic data port. Data ports are used for modeling data communication among Circuits or Assemblies. PortData is an abstract element. *PortDataIn* and *PortDataOut* specialize PortData and represent input and output data port, respectively. PortDataOut has a on-to-many relationship with PortDataIn, *dataOutToDataIn*, meaning that a value on the data output port can be fed to several input ports.

3.3 Triggering elements

Figure 3(b) shows a metamodel fragment containing the architectural elements for modeling the control flow. *PortTrig* models a generic trigger port. *PortTrigIn* and *PortTrigOut* specialize PortTrig; they represent trigger input and output ports, respectively. Trigger ports are used for specifying precedence and control over the architectural elements. A trigger output port generates trigger signal upon the completion of the related Circuit. When receiving a trigger signal, a trigger input port triggers the execution of the related Circuit's Behavior. Any trigger signal after the first received is meaningless, therefore ignored. PortTrig, PortTrigIn and PortTrigOut are abstract. PortTrigIn is specialized by *PortTrigInMode*, *PortTrigInAssembly* and *PortTrigInInterface*. Similarly, PortTrigOut is specialized by *PortTrigOutMode*, *PortTrigOutAssembly* and *PortTrigOutInterface*. A Mode is composed by, at least, one PortTrigInMode and one PortTrigOutMode; PortTrigOutMode has a one-to-many relationship with PortTrigInMode,



(a) Metamodel fragment for the data flow objects



(b) Metamodel fragment for the control flow objects

Fig. 3. RCM metamodel fragments

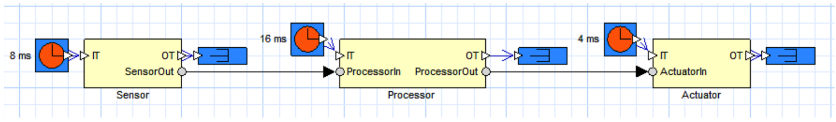
meaning that a trigger output port can trigger more than one trigger input port. Similarly, Interface is composed by exactly one PortTrigInInterface and, at least, one PortTrigOutInterface. Also, PortTrigOutInterface has a one-to-many relationship with PortTrigInInterface. Finally, an Assembly might contain PortTrigInAssembly and PortTrigOutAssembly, where a PortTrigOutAssembly has a one-to-many relationship with PortTrigInAssembly.

4 RCM2AUTOSAR transformation

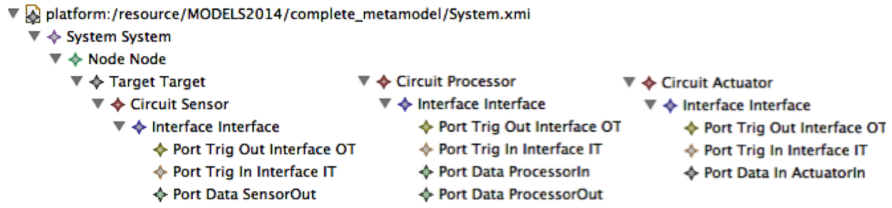
In this section, we describe the RCM2AUTOSAR transformation. Figure 4(a) depicts a RCM model of a single node real time system composed by three Circuits, *Sensor*, *Processor* and *Actuator*.

Sensor has one trigger input port, *IT*, one trigger output port, *OT*, and one data output port, *SensorOut*. Similarly, Processor has two trigger ports and two data ports, and Actuator has two trigger ports and a data port. Intuitively, the

model describes a vehicle function in which data are sensed, processed and fed to the actuator for the stimulation [3].



(a) RCM model of a single-node real time system



(b) RCM model serialization

Fig. 4. RCM model and its serialization

Although trivial, the transformation is used for miming typical scenarios in the software development of vehicle embedded systems. With models as that depicted in Figure 4(a), manual translations might still appear feasible; nevertheless, in reality, vehicle embedded systems are composed by over 70 embedded systems and thousands components [14]. Also, the transformation is used for proving the validity of the metamodel introduced in the Section 3. Figure 4(b) shows the textual serialization of the model.

Algorithm 1 shows the metacode for the RCM2AUTOSAR transformation. The algorithm mainly consists of two relationships between RCM and AUTOSAR elements, which are: Circuit to Software Component, and PortData to PortClientServer [4]. The former relationship exploits a naming convention for better translating the elements avoiding flattening the RCM model. The two involved metamodels do not have the same expressiveness, which means that the underneath relationship is partial. Indeed the are some elements of RCM which are ignored from the transformation. Figure 5 shows the serialization of the AUTOSAR model obtained as a result of the transformation.

5 Related Works

The embedded system research community and the vehicular industry have focused more and more on the definition of component-based technologies for embedded vehicular systems. Hereafter, we present and discuss some attempts targeted towards the development of resource-constrained vehicular real-time systems.

[3] The model contains triggering elements not presented in this work, i.e., clock and trigger terminator elements. The reader can assume they are responsible for triggering the circuits and terminating the control flow, respectively.

[4] The explanation of the AUTOSAR metamodel is outside the foucs of this work. The interested reader may refer to the [1].

Algorithm 1 RCM2AUTOSAR transformation

```
1: new VirtualFunctionBus VFB;
2: for each Circuit c in a Target t do
3:   switch c.name do
4:     case (1) //c.name ends in Sensor
5:       new SensorSoftwareComponent sc;
6:       sc.name = c.name;
7:     case (2) //c.name ends in Actuator
8:       new ActuatorSoftwareComponent sc;
9:       sc.name = c.name;
10:    case (default)
11:      new SoftwareComponent sc;
12:      sc.name = c.name;
13:  for each Interface i in c do
14:    for each PortDataIn di in i do
15:      new RequiredPortClientServer rp;
16:      rp.name = di.name;
17:    end for
18:    for each PortDataOut do in i do
19:      new ProvidedPortClientServer pp;
20:      pp.name = do.name;
21:      pp.receiver = do.dataOutToDataIn;
22:    end for
23:  end for
24: end for
```

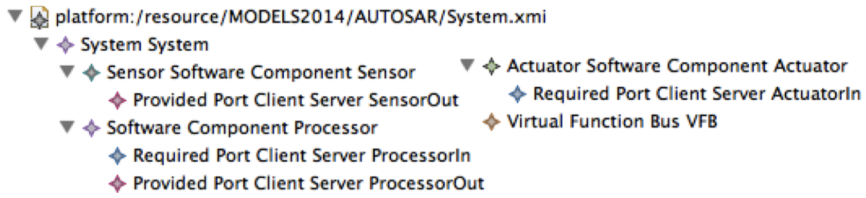


Fig. 5. Serialization of the obtained AUTOSAR model

5.1 EAST-ADL/AUTOSAR

AUTOSAR [1] is an industrial initiative to provide standardized software architecture for the development of embedded software for the vehicular domain. Within AUTOSAR, the software architecture is defined in terms of Software Components (SWCs) and Virtual Function Bus (VFB). VFB handles the virtual integration and communication among SWCs, hiding the low-level implementation details. Compared with RCM, EAST-ADL/AUTOSAR describes the software at a higher level of abstraction. It has no ability to specify and handle timing information at design time, such as component worst case execution time. AUTOSAR does not distinguish between data and control flow, as well as between inter and intra node communication. Contrariwise, RCM was specifically designed taking timing requirements into account. As shown in Section 3, RCM

clearly separates data and control flow; also, it has been recently extended with special network interface components for modeling inter-node communication [2]. The AUTOSAR sender receiver communication mechanism is very similar to the RCM pipe-and-filter communication mechanism. In short, AUTOSAR focuses on hiding the information which RCM highlights.

5.2 TIMMO/TIMMO-2-USE

TIMMO [3] is a large EU research project, which aims to provide AUTOSAR with a timing model [4]. To this end, it provides a predictable methodology and language TADL [5] for expressing timing requirements and constraints. TADL is inspired by MARTE [6], which is an UML profile for model-driven development of real-time and embedded systems. The TIMMO predictable methodology makes use of the EAST-ADL and AUTOSAR interplay, where the former is used for the software structural modeling, while the latter is used for the implementation. Although the TIMMO project has been evaluated upon prototype validators, from the best of our knowledge, there is no concrete industrial implementation of the TIMMO project. TIMMO-2-USE [7] follows-up on the TIMMO project. It presents a major redefinition of TADL and new functionality for supporting the AUTOSAR extensions regarding timing model. Arcticus Systems has been involved in TIMMO-2-USE project as one of the industrial partners. Both TIMMO and TIMMO-2-USE attempt to annotate AUTOSAR with a timing model. This may be hard to accomplish as AUTOSAR aims at hiding implementation details of execution environment and communication using the Virtual Function Bus, as shown in Section 4. That is, at the modeling level, there is no information in AUTOSAR to express low level details, e.g., linking information, which is necessary to extract the timing model from the software architecture. There is no focus in these initiatives on how to extract this information from the model or perform timing analysis or synthesize the run-time framework.

5.3 ProCom

ProCom [8] is a two-layered component model for the development of distributed embedded systems. It is the result of a research project conducted at Mälardalen University. The upper layer, ProSys, models the system and concurrent subsystems communicating by means of asynchronous messages. The lower layer, ProSave, models each subsystem in terms of functional components implemented as a piece of code. Being inspired by RCM, ProCom presents several similarities with it. Both CMs have passive components, clearly separate the control flow from the data flow and use the pipe-and-filter communication mechanism for components interconnection. However, ProCom does not differentiate between intra- and inter-node communication which is unlike RCM. As for AUTOSAR, also ProCom hides communication details, making hard the extraction of timing model and the execution of timing analysis [12].

6 Conclusions and Future Works

In the last decades, CBSE has enhanced the software development for vehicular embedded systems. Nevertheless, industry needs to move further towards a

seamless development chain for reducing software development costs and time-to-market. In this respect, one of the major challenge is the definition of a methodology and accompanying technologies. In this work we proposed the adoption of a methodology exploiting the crossplay of MDE and CBSE and took initial steps towards the realization of the aforesaid seamless chain. We i) motivated the usage of RCM within the vehicular domain, by highlighting its unique features against existing CMs, ii) formalized a metamodel based on RCM and ii) proved the metamodel validity by means of the RCM2AUTOSAR model transformation. The formalization of the metamodel not only serves as base for embracing the MDE vision, but it also aims in restoring the separation of concerns which has been lost during the evolution of the RCM. For sake of space we omitted a comparison between RCM and its metamodel. As future works, we will investigate further metamodel refinements targeting the enhancement of vehicular tool chaining while preserving the current expressive power. The RCM2AUTOSAR transformation outlines the potential benefits gained in having a proper metamodel for RCM, in terms of automation, interoperability and traceability. As future investigation direction we will also, together with our industrial partners, cover the identification of additional languages used along the software development for the vehicular embedded systems, with the aim of formalizing their metamodels and hence enable model transformations for supporting a more extensive tool chain.

Acknowledgment

The work in this paper is supported by the Swedish Knowledge Foundation (KKS) within the project FEMMVA and Swedish Research Council (VR) within the project SynthSoft. We thank our industrial partners Arcticus Systems and Volvo Construction Equipment, Sweden.

References

1. AUTOSAR Techincal Overview `http://autosar.org`
2. Mubeen, S., Mäki-Turja, J., Sjödin, M.: Communications-Oriented Development of Component-Based Vehicular Distributed Real-Time Embedded Systems. Journal of Systems Architecture. vol. 60, pp. 207-220 (2014)
3. TIMMO Methodology. DELIVERABLE 7 (2009)
4. Mastering Timing Information for Advanced Automotive Systems Engineering (2012)
5. TADL: Timing Augmented Description Language. Deliverable 6 (2009)
6. The UML Profile for MARTE: Modeling and Analysis of Real-Time and Embedded Systems (2010)
7. TIMMO-2-USE `http://www.timmo-2-use.org/`
8. Sentilles, S., Vulgarakis, A., Bures, T., Carlson, J., Crnkovic, I.: A Component Model for Control-Intensive Distributed Embedded Systems. In: 11th International Symposium on Component Based Software Engineering, pp. 310-317 (2008)
9. EAST-ADL Domain Model Specification. V 2.1.12 (2013)
10. Rubus models, methods and tools `http://www.arcticus-systems.com`
11. Hänninen, K., et.al.: The Rubus Component Model for Resource Constrained Real-Time Systems. In: 3rd IEEE International Symposium on Industrial Embedded Systems (2008)

12. 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, no. 1, pp. 453-482 (2013)
13. 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).
14. Charette, R. N.: This Car Runs On Code. In: *Spectrum, IEE*, vol. 46, no. 2 (2009)
15. Crnkovic, I.: Component-Based Software Engineering for Embedded Systems. In: 27th International Conference on Software Engineering, pp. 712-713 (2005)
16. Bézivin, J.: On the unification power of models. Software & Systems Modeling, vol. 4, no. 2, pp. 171-188 (2005)
17. Kent, S.: Model Driven Engineering. Lecture Notes in Computer Science. vol. 2335, pp.286-298 (2002)