IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 12, 2016, accepted October 20, 2016, date of publication December 16, 2016, date of current version June 28, 2017.

Digital Object Identifier 10.1109/ACCESS.2016.2641218

A Metamodel for the Rubus Component Model: Extensions for Timing and Model Transformation From EAST-ADL

ALESSIO BUCAIONI[1,2], (Member, IEEE), ANTONIO CICCHETTI[1], (Senior Member, IEEE), FEDERICO CICCOZZI[1], (Senior Member, IEEE), SAAD MUBEEN[1], (Senior Member, IEEE), AND MIKAEL SJÖDIN[1],

[1]School of Innovation, Design and Engineering, Mälardalen University, Västerås 72123, Sweden
[2]Arcticus Systems AB, Järfälla, Sweden

Corresponding author: A. Bucaioni (alessio.bucaioni@mdh.se)

This work was supported in part by the Swedish Knowledge Foundation through the SMARTCore Project, in part by the Swedish Research Council through the SynthSoft Project, and in part by the Swedish Foundation for Strategic Research through the PRESS Project.

ABSTRACT According to the model-driven engineering paradigm, one of the entry requirements when realizing a seamless tool chain for the development of software is the definition of metamodels, to regulate the specification of models, and model transformations, for automating manipulations of models. In this context, we present a metamodel definition for the Rubus component model, an industrial solution used for the development of vehicular embedded systems. The metamodel includes the definition of structural elements as well as elements for describing timing information. In order to show how, using model-driven engineering, the integration between different modeling levels can be automated, we present a model-to-model transformation between models conforming to EAST-ADL and models described by means of the Rubus component model. To validate our solution, we exploit a set of industrial automotive applications to show the applicability of both the Rubus component model metamodel and the model transformation.

INDEX TERMS Computer applications, computer aided analysis, computer aided engineering, embedded software, software systems, software engineering, software architecture, model-driven development vehicles, metamodeling, system modeling language.

I. INTRODUCTION

During the last decades, industrial demands on vehicular embedded systems have constantly grown causing an increment in complexity of the related software. It has been estimated that current vehicles can have more than 70 embedded systems running up to 100 million lines of code [1]. On the one hand, industry needs efficient processes to cope with the size of these systems for optimising software development cost and time-to-market. On the other hand, most vehicular embedded systems have extra-functional requirements that have to be taken into account from the early stages of the development. More specifically, vehicular embedded systems are real-time systems [2], meaning that they must deliver their functionality within their timing deadlines. Consequently, timing requirements are crucial for these systems.

Lately, Model-Driven Engineering (MDE) has gained both academic and industrial recognition as an effective practice for dealing with the increasing complexity of modern embedded software [3]. MDE [4] is an engineering paradigm that addresses software development as the process of (i) designing models and (ii) refining them, starting from higher and moving towards lower levels of abstraction, via the so-called model transformations. Moreover, it allows to cope with extra-functional properties, e.g., timing properties, by annotating the models with properties and constraints, e.g., Worst Case Execution Time (WCET), thus enabling model-based analysis, e.g., end-to-end response time and delay analysis [5]. AUTOSAR [6] and the Rubus Component Model (RCM) [7], to name a few, are examples of well-known and established solutions used within the vehicular domain. Lately, AUTOSAR has been complemented with the EAST-ADL methodology [8]. EAST-ADL is an architectural description language which provides concepts and methods for managing and organising the various artefacts produced along the software development of vehicular embedded systems [9]. It promotes the separation of concerns through a top-down software development process relying on four different abstraction levels, i.e., vehicle, analysis, design and implementation. In the latter, EAST-ADL makes use of AUTOSAR. While EAST-ADL has been proven successful in

coping with the complexity and size of vehicular embedded software, it still provides limited support for dealing with timing requirements. In fact, by employing AUTOSAR at the implementation level, most of the timing, implementation and communication details are hidden by the so-called Virtual Function Bus (VFB). This information is necessary for verifying timing requirements. For this reason, an increasing number of vehicular manufacturers (e.g., Volvo CE, BAE Systems) is using RCM as an alternative to AUTOSAR.

Since heterogeneous languages are used in the development process (e.g., EAST-ADL, AUTOSAR, RCM), in order to allow a smooth interplay between them, proper automated mechanisms are needed for the translation among the various artefacts specified using the various languages. Manual mechanisms, in fact, are not only tedious, time consuming and error-prone, but even infeasible in most cases due to the size and complexity of industrial artefacts. From a broader perspective, interoperability among languages is a key factor within the software development, as acknowledged by several international projects.[1]

In this paper, we leverage MDE for automating the transition between EAST-ADL and RCM with the aim of reducing software development cost and time to market [10]. To this end, in order to embrace the MDE paradigm and benefit from its features, we propose (i) a metamodel definition for RCM (called RubusMM in the remainder of this paper) and (ii) a model-to-model transformation between EAST-ADL and RCM (DL2RCM). We define RubusMM bearing in mind the following aspects:

backward compatibility: the metamodel should not hinder the migration of legacy RCM artefacts;

extensibility: the metamodel should disclose the opportunity to integrate, in a smooth way the RCM modelling environment, in a typical automotive development chain.

Finally, we leverage an industrial automotive application for showing the usability and applicability of RubusMM and DL2RCM.

The rest of the paper is organised as follows. Section II presents the context of this work together with its related works. Section III introduces RubusMM and its extensions for timing elements. Section IV shows the DL2RCM transformation while Section V discusses its applicability on a case study. Finally, Section VI provides details on the validation of our solution and highlights the benefits of having a metamodel definition for RCM while Section VII draws conclusions and discusses future works.

II. BACKGROUND AND RELATED WORK

In this section, we describe the context of this research and the related works. In Section II-A we discuss the use of MDE and CBSE paradigms in the vehicular domain. In Section II-B we describe timing analyses and timing models while in Section II-C we discuss the paper contributions in relation with authors' previous works.

A. MDE AND CBSE IN THE AUTOMOTIVE DOMAIN

MDE is a paradigm which aims at raising the level of abstraction of software development by focusing on modelling activities rather than coding. In this context, MDE promotes *models* and *model transformations* as first-class citizens. Models represent an abstraction of the system under development, from a particular point of view [11]. The set of rules and constraints needed for building a valid model is specified in the so-called *metamodel*. Formally, a metamodel defines the abstract syntax of a well-formed model; the relation between metamodel and models is called conformance. Model transformations represent the means of refinement by which models are manipulated [12]. In fact, model transformations translate a source model into a target model keeping their conformance to the respective metalled intact. According to the MDE paradigm, starting from a model and by means of model transformations it is possible to automatically obtain a variety of artefacts, such as new models, code, etc. In this context, the entire software development can be seen as a transformation process where low level abstraction models are automatically obtained by transforming higher-level abstraction models. Within the automotive domain, the adoption of MDE and CBSE paradigms led to the standardisation of an architectural description language, namely EAST-ADL [8]. EAST-ADL proposes a view over the development process composed by four different abstraction levels. Figure 1 shows the abstraction levels together with methodologies and languages used at each one of them.

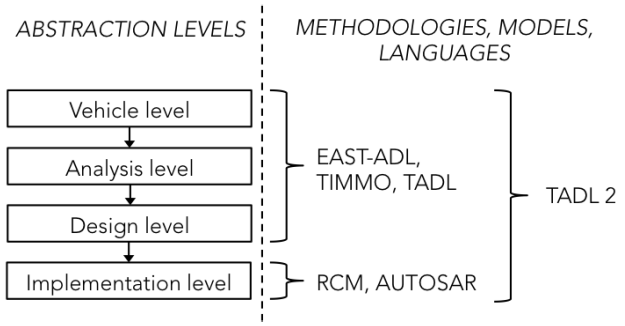

FIGURE 1. EAST-ADL abstraction levels.

The *vehicle level* is the highest abstraction level and captures information regarding the system's functionality. At this level, feature models can be used for showing what the system provides in terms of functionality. In addition, these models are decorated with requirements. The vehicle level is also known as end-to-end level as it serves to capture requirements and features on the end-to-end vehicle functionality. At the *analysis level*, vehicle functions are expressed, using formal notations, in terms of behaviours and interfaces. Yet, design and implementation details are omitted. The artefact developed at this level is called Functional Analysis Architecture.

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

At this stage, high level analysis for functional verification can be performed. At the *design level*, the analysis-level artefacts are refined with design-oriented details: while the analysis level does not differentiate among software, middleware and hardware, the design level explicitly separates them. Allocation of software functions to hardware nodes is expressed at this level too. The artefacts developed at this level include Functional Design Architecture and Hardware Design Architecture. At the *implementation level*, artefacts introduced at the design level are refined with implementation details. The output of this level is a complete software architecture which can be used for code generation. At this stage component models as RCM and AUTOSAR, are used to model the system in terms of components and interactions among them.

AUTOSAR is an industrial initiative to provide standardised software architecture for the software development of vehicular embedded systems. Within AUTOSAR, the software architecture is defined in terms of AUTOSAR *software components* (AutosarSWCs) and *VFB*. VFB is a black box component which handles the virtual integration and communication among AutosarSWCs, hiding low-level implementation details [13]. AUTOSAR describes the software at a high level of abstraction focusing on the functional and structural aspects of the architecture. Also, it does not distinguish between data and control flow, as well as between inter- and intra-node communication. Lately, AUTOSAR has been provided with a timing model within the two EU research projects TIMMO [14] and TIMMO-2-USE [15], respectively. To this end, TIMMO provides a predictable methodology and language, called TADL [16] for expressing timing requirements and constraints. TADL is inspired by MARTE [17], which is a UML profile for modelling and analysis of real-time embedded systems. The TIMMO methodology makes use of EAST-ADL and AUTOSAR interplay, where the former is used for the software functional modelling at higher abstraction levels, while the latter is used for the modelling of software architecture and execution information at the implementation level. TIMMO-2-USE [15], a follow up project, presents a major redefinition of TADL in TADL2 [18]. The purpose of this project is to include new functionality for supporting the AUTOSAR extensions regarding timing model. Although both TIMMO and TIMMO-2-USE attempt to annotate AUTOSAR with a timing model, AUTOSAR is still not expressive enough for representing timing, implementation and communication information of the software architecture as this information is hidden by VFB. In this context, an increasing number of vehicular manufacturers, e.g., Volvo CE, BAE systems, prefers RCM to AUTOSAR.

RCM is developed by Arcticus Systems in collaboration with Mälardalen University and it is used for model- and component-based development of resource-constrained embedded real-time systems. The main goal of RCM is to express the software architecture in terms of software functions and interactions among them. In RCM, the basic entity is the so-called *software circuit* (SWC) which represents the lowest-level hierarchical element in RCM and encapsulates basic software functions. Each SWC is defined by its *behaviour* and *interface*. Interfaces manage the interactions among SWCs via ports. RCM distinguishes between data and control flow. Therefore, the interfaces have two types of ports: *data ports* for the data flow and *trigger ports* for the control flow, respectively. The SWC is characterised by run-to-completion semantics, meaning that, upon triggering, it reads data from the data input ports, executes its behaviour and writes data on the data output ports. SWCs can be grouped and organised in *assemblies*, decomposing the system in a hierarchical manner. *Modes* are used to represent different configurations of the software architecture. *Target* entities are used for grouping modes that are deployed on the same Electronic Control Unit (ECU). Moreover, they provide details regarding hardware and operating system. *Node* is a hardware and operating-system independent abstraction of a target entity. Finally, *System* is the top-level hierarchical entity, which describes software architecture for the complete vehicular system. RCM facilitates analysis and reuse of components in different contexts by separating functional code from the infrastructure that implements the execution environment. Compared to AUTOSAR, RCM allows the developer to specify and handle timing information at design time. It also distinguishes between data and control flow as well as inter- and intra-node communication. To this end, RCM has been recently extended with special network interface components for modelling the inter-node communication [19]. The RCM pipe-and-filter communication mechanism is very similar to the AUTOSAR sender-receiver communication mechanism. In short, RCM was specifically designed for expressing all the low-level information needed for extracting the timing model from the software architecture.

B. END-TO-END TIMING MODELS AND ANALYSES

End-to-end timing analysis is a key activity for the verification and validation of vehicular real-time systems. Therefore, a tool chain that is used for the model- and component-based development of vehicular systems shall support such an analysis. To support it an appropriate system view, called end-to-end timing model, should be extracted from the software architecture. In particular, an end-to-end timing model comprises timing properties, requirements, dependencies and linking information concerning all tasks, messages and task chains in a distributed embedded system under analysis.[2] An end-to-end timing model is composed of two models namely a timing model and a linking model. In order to elaborate this, consider a task chain distributed over three nodes connected by a network as shown in Figure 2.

The system timing model captures all the timing information about the three nodes and the network. Whereas the linking model includes all the linking information in the task chains, including the control and data flows. The analysis engine uses these models for performing end-to-end

[2]We refer the reader to [19] for details about the timing model.

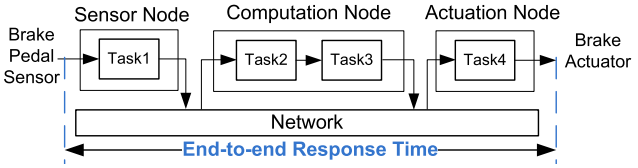

FIGURE 2. Example showing end-to-end response time.

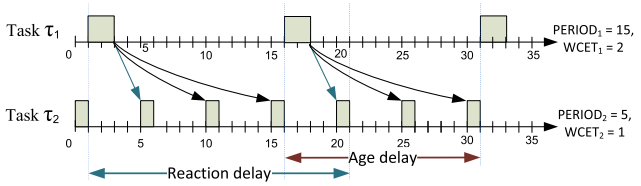

FIGURE 4. Example showing end-to-end delays.

timing analyses. We refer the reader to [5] for further details about the end-to-end timing analysis. The analysis results consist of response times of tasks and messages as well as system utilisation. Also, the analysis calculates end-to-end response times and delays.

The end-to-end response time of a task chain is equal to the elapsed time between the arrival of a stimulus, e.g., the brake pedal sensor input in the sensor node, and the response to it, e.g., the brake actuation signal in the actuation node as shown in Figure 2. Within a task chain, if the tasks are triggered by independent sources, then it is important to calculate different types of delays such as age and reaction.

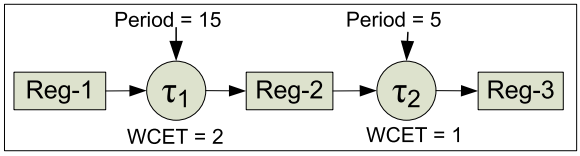

FIGURE 3. A task chain with independent activations of tasks.

An *age delay* corresponds to the freshness of data. It finds its importance in control systems used in the vehicles. Whereas, the *reaction delay* corresponds to the first reaction for a given stimulus. This delay finds its application in body electronics in the vehicles. In order to explain these delays, consider a task chain in a single-node system as shown in Figure 3. There are two tasks in the chain denoted by τ_1 and τ_2. The tasks are triggered by independent clocks of periods 15ms and 5ms, respectively. Let the WCETs of these tasks be 2ms and 1ms, respectively. τ_1 reads data from the register Reg-1 and writes data to Reg-2. Similarly, τ_2 reads data from the Reg-2 and writes data to Reg-3. Since the tasks are activated independently by different clocks, there can be multiple outputs (Reg-3) corresponding to one input (Reg-1) to the chain as shown by several uni-directional arrows in Figure 4. The age and reaction delays are depicted in Figure 4. The age delay is equal to the time elapsed between the current non-overwritten release of τ_1 and corresponding last response of τ_2 among all valid data paths. Whereas, the reaction delay is equal to the time elapsed between the previous non-overwritten release of τ_1 and the first response of τ_2 corresponding to the current non-overwritten release of τ_1. These delays are equally important in distributed embedded systems.

We consider the end-to-end timing model that corresponds to the holistic schedulability analysis for distributed embedded systems [20]. Stappert *et al.* [21] described end-to-end timing constrains for multi-rate automotive embedded systems. In [22], Feiertag *et al.* presented a framework (developed as part of the TIMMO project) for the calculations of end-to-end delays. A scalable technique, based on model checking, for the computation of end-to-end latencies is described in [23].

C. PAPER CONTRIBUTIONS AND RELATION WITH AUTHORS' PREVIOUS WORK

This paper extends our previous work [24] where we presented a preliminary metamodel definition of the RCM core elements only (the so-called *backbone*). There, we discussed the general benefits of having a metamodel definition for RCM rather than making EAST-ADL and RCM to interoperate. The metamodel presented in this paper extends the previous one with 13 new metaclasses, which are needed for modelling control and data flows as well as for the specification of timing properties and constraints for different types of delay for single-node and distributed embedded systems. While the initial version of the metamodel, presented in [24], was useful for brainstorming and running rudimental experiments in terms of modelling of static definitions of simple applications, the extensions to the metamodel presented in this paper enable its use in practice for both modelling and analysis purposes in real-world use cases. In [25], we discussed the idea of translating timing constraints from EAST-ADL to RCM. However, that work did not provide any support for either the (meta-)modelling definition of these constraints nor any automation in terms of model-to-model transformation for the translation of constraints from EAST-ADL to RCM. Moreover, in this paper we present – for the first time – the DL2RCM model-to-model transformation which enables the automatic translation of artefacts specified using EAST-ADL to artefacts specified using RubusMM. Additionally, we discuss the applicability of RubusMM and DL2RCM by leveraging an industrial automotive application and we validate RubusMM's expressiveness by exploiting several real-life automotive models.

III. A METAMODEL DEFINITION FOR THE RUBUS COMPONENT MODEL

In this section, we describe RubusMM (a metamodel definition for RCM) by comparing it with the previous metamodel definition given in [24], thus highlighting differences and commonalities. For the sake of readability, we present the metamodel in four fragments; however, these fragments can

be combined by matching metaclasses with the same names and this combination represents the complete RubusMM. Figure 12 in section A shows the metamodel's backbone. The top metaclass is *System*, which acts as a container for the whole architecture. System, as all the metaclasses in the metamodel, inherits from the abstract metaclass *NamedElement*. A System element contains one or more elements of type *Node*. A Node element is an abstraction of a *Target* element independent of hardware and operating-system; it groups all the software architecture elements which realise a specific function. Its reference *activeTarget* defines which target, among those specified, is active for a certain node. Target is a hardware and operating-system specific instance of a Node; it serves for modelling the deployment of the software architecture. This means that it contains all the functions to be deployed on the same ECU. Consequently, a Node can be realised by different Target elements, depending on the hardware and the operating system used for the deployment, for example, PowerPC with Rubus Operating System, Simulated target with Windows operating system. A Target element contains one or more elements of type *Mode*. A Mode represents a specific application of the software architecture as, for instance, start-up or low-power mode. A Mode element might contain elements of type *Circuit* and *Assembly*. A Circuit is the lowest-level hierarchical element which encapsulates basic functions. It contains an element of type *Interface* and one or more elements of type *Behavior*. An Interface groups all the data and trigger ports of a certain circuit while a Behavior contains the code to be executed by the specific Circuit. The reference *activeBehavior* is used for specifying which behaviour is active for the related circuit. Assembly elements do not add any semantics to the architecture: they are used for grouping and organising circuits and assemblies in a hierarchical fashion. Figure 5 depicts a RCM model consisting of a circuit element Circuit containing a behavior element Behavior and an interface element Interface. In turns, Interface contains two trigger ports namely PotTrigIn and PotTrigOut. A *Connector* realises the actual communication between two ports. *ConnectorData* and *ConnectorTrig* metaclasses are used for modelling the communication between data ports and control ports, respectively. RCM explicitly separates data and control flow. Both ConnectorData and ConnectorTrig metaclasses inherit from the abstract metaclass Connector. Please note that all the metaclasses in this fragment were part of the previous metamodel definition.

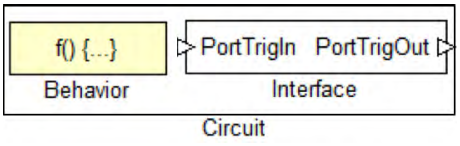

FIGURE 5. RCM model consisting of circuit, behavior, interface and control port elements.

Figure 13 in section A shows the metamodel fragment containing the concepts used for modelling the data flow. The abstract metaclass *PortData* models a generic data port. It has three attributes: *dataPassingMethod* specifies how data is passed to the port, *dimension* expresses the size of the port while *initialValue* specifies its initial value. The metaclass PortData is specialised by the metaclasses *PortDataIn* and *PortDataOut*, which model an input and output data port, respectively. They are contained in the Interface and the Assembly metaclasses for modelling the data communication among circuits and assemblies, respectively. Figure 11 depicts PortData elements as white circles on the border of the circuit elements. As aforesaid, the metaclass ConnectorData is used for modelling the actual communication between two data ports. In this respect, the references *sourcePort* and *targetPort* are used for specifying the ports involved in the communication. Figure 11 also depicts ConnectorData elements as black arrows among PortData elements. Please note that the metaclasses *Connector*, *ConnectorData* and *dataPassingMethod* were not part of the previous definition of the metamodel. Adding the aforesaid three metaclasses, give us the possibility to explicitly model the data connection among circuit elements.

Figure 14 in section A shows the metamodel fragment containing the concepts that can be used to represent the control flow. The metaclasses *PortTrigIn* and *PortTrigOut* describe an input trigger port and an output trigger port, respectively. They both inherit from the metaclass *PortTrig*, which describes a generic trigger port. Modes, assemblies and interfaces are composed of input and output trigger ports for modelling the control flow among modes, assemblies and circuits, respectively. In Figure 6, the trigger ports of the mode elements Mode1 and Mode2 are represented as white triangles on the border of the mode elements. The *ConnectorTrig* metaclass inherits from the abstract metaclass Connector. It has two references, *sourcePort* and *targetPort*, used for modelling the actual communication between trigger ports. Figure 6 depicts three ConnectorTrig elements as blue arrows linking the trigger ports. *Clock* and *Sink* elements are responsible to start and end the execution of a software circuit, respectively. Clock elements have a *Period* attribute for expressing the period of the clock in milliseconds. Figure 11 depicts Clock and Sink elements as blue boxes with a red clock and blue boxes, respectively. Please note that the metaclasses *Connector*, *ConnectorTrig*, *Clock* and *Sink* were not part of the previous definition of the metamodel. Also, the hierarchy of trigger ports has been substantially streamlined passing from 9 metaclasses of the previous metamodel definition to the 3 of the current one.

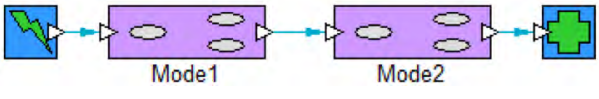

FIGURE 6. RCM model consisting of mode, connector and control port elements.

Figure 15 in section A depicts an excerpt of the RCM metamodel containing the metalements representing timing

TABLE 1. Main relations holding in the DL2RCM transformation.

EAST-ADL elements	RCM elements	Conditions/Assumptions
Model	(hierarchy of) System, Node, Target, Mode	If we consider all the EAST-ADL abstraction levels then the hierarchy is not necessary
DesignFunctionPrototype	Assembly	If the associated DesignFunctionType is not elementary
DesignFunctionPrototype	(hierarchy of) Circuit, Interface	If the associated DesignFunctionType is not elementary
FunctionConnector	ConnectorData	
FunctionFlowPort	PortData	
AgeConstraint	DataAgeStart, DataAgeEnd	
ReactionConstraint	DataReactionStart, DataReactionEnd	
	ConnectorTrig, PortTrig, Clock, Sink	EAST-ADL does not explicitly model the control flow. Therefore these RCM elements are deducted considering the whole architecture

constraints and properties for different types of delays in event chains. The notion of different delay types is meaningful in multi-rate systems where components in the event chain can be triggered with independent clocks. Hence, there can be multiple occurrences of response corresponding to a single occurrence of stimulus in the chain. In RCM, these constraints are specified by means of two model elements placed at the beginning and at the end of the event chain. The metaclasses which represent the data reaction constraint are *DataReactionStart* and *DataReactionEnd*, while the metaclasses which model the data age constraint are the *DataAgeStart* and *DataAgeEnd*. DataAgeStart and DataReactionStart inherit from the abstract metaclass *DataStart*, while DataAgeEnd and DataReactionEnd inherit from the abstract metaclass *DataEnd*. The *deadline* attribute of the DataEnd metaclass specifies the maximum value for the related reaction along the enclosed chain expressed in milliseconds. DataStart and DataEnd inherit from the abstract metaclass *Data*, which models a generic delay constraint. It contains a data input port and a data output port, meaning that the data traveling along the data chain must traverse the delay constraint for activating it. Figure 11 depicts the aforesaid delay constraints as grey squares along the circuit chains. Please note that the whole timing fragment was not part of the previous definition of the metamodel.

Please note that the complete specifications of RCM and RubusMM are not publicly available. Arcticus Systems AB, in fact, remains the only specifications owner. However, the interested reader might refer to [7] for checking the completeness of RubusMM with respect to RCM. Moreover, the RubusMM was developed together with Arcticus Systems AB for ensuring its adherence to RCM.

IV. DL2RCM MODEL TRANSFORMATION

In this section we present DL2RCM, a model-to-model transformation from the EAST-ADL Design Level metamodel to RubusMM. The intent is to show how, having a proper metamodel for RCM, it is possible to realise a seamless tool chain and complement EAST-ADL with the RCM's timing analysis capabilities. In Section II, we showed how the EAST-ADL methodology (EAST-ADL complemented by AUTOSAR at the implementation level) uses the four abstraction levels for implementing a top-down development process. In this respect, we presented RCM and AUTOSAR to be technologies used at the bottom abstraction level, i.e., implementation level. In our previous work we proposed RCM as an alternative to AUTOSAR within an EAST-ADL development methodology. To this end, we believe it is crucial to show that RCM fully integrates within the EAST-ADL methodology. That is, considering the EAST-ADL four abstraction levels, it is possible to synthesise an EAST-ADL Design Level model to a corresponding RCM model. The DL2RCM transformation is used for performing such an integration automatically. The benefits of realising this in an automatic manner become more visible when considering that the involved technologies, EAST-ADL and RCM, are used for representing complex architectures, for which manual translations are not only tedious, time consuming, and error-prone, but they might even become unfeasible. The DL2RCM transformation is a unidirectional model-to-model transformation from the EAST-ADL Design Level metamodel to RubusMM. The latter has been presented in Section III. The former has been described in [8] and implemented as a UML profile within the Eclipse Papyrus project.[3] Figure 16 in section A shows the except of the EAST-ADL metamodel containing the concepts involved by the DL2RCM transformation, [4] thus the concepts from the EAST-ADL FunctionalModeling and TimingContraints [5] packages. The relation underneath the

[3] http://eclipse.org/papyrus/

[4] The explanation of theEAST-ADL metamodel is outside the scope of this work. The interested reader may refer to [8].

[5] Timing constraints, occurrences and events are part of the TADL2 [18] language. Starting from the V2.1.11 release, EAST-ADL incorporates TADL2 language in its specification.

Algorithm 1 DL2RCM Transformation

```
1:  function MODEL2SYSTEM(Model m)
2:      Mode mo = createHierarchy(m);
3:      FDP(m.functionalDesignPrototype, mo)
4:      TC2TC(fdp, mo)
5:  end function
6:
7:  function FDP(DesignFunctionPrototype fdp, Mode mo)
8:      if fdp is not elementary then
9:          Assembly a = createAssembly(fdp, mo);
10:         for connector in fdp do
11:             c2c(connector, a)
12:         end for
13:         for part in fdp do
14:             if part is not elementary then
15:                 Assembly as = DP2A(part, a);
16:             else
17:                 Circuit ci = DP2C(part, a);
18:             end if
19:         end for
20:     else
21:         Circuit c = createCircuit(fdp, mo);
22:     end if
23: end function
```

Algorithm 1 DL2RCM Transformation

```
24: function C2C(FunctionConnector fc, Assembly a)
25:     ConnectorData con = createConnectorData(fc, a);
26:     for end in fc do
27:         if end.functionPrototype is not elementary then
28:             Assembly as = DP2A(end.functionPrototype, a);
29:             if end.functionPort is FunctionFlowPort then
30:                 if end.functionPort is in then
31:                     ConnectorTrig conTC = createControlFlowIn(fc, a);
32:                     PortDataIn pdi = CreatePortDataIn(fc.functionPort, as);
33:                 else
34:                     ConnectorTrig conTS = createControlFlowOut(fc, a);
35:                     PortDataOut pdo = CreatePortDataOut(fc.functionPort, as);
36:                 end if
37:             else
38:             end if
39:         else
40:             Circuit c = DP2C(e.functionPrototype, a);
41:             if end.functionPort is FunctionFlowPort then
42:                 if end.functionPort is in then
43:                     ConnectorTrig conTC = createControlFlowIn(fc, a);
44:                     PortDataIn pdi = CreatePortDataIn(fc.functionPort, c);
45:                 else
46:                     ConnectorTrig conTS = createControlFlowOut(fc, a);
47:                     PortDataOut pdo = CreatePortDataOut(fc.functionPort, c);
48:                 end if
49:             else
50:             end if
51:         end if
52:     end for
53: end function
```

transformation is non-bijective meaning that there is not a one-to-one mapping between the elements involved in the transformation. In this respect, in order to preserve as much information as possible, assumptions are needed when defining the relations composing the transformations. Table 1 summarises these assumptions [6] together with the involved EAST-ADL and RubusMM elements. Please note that the intent of the DL2RCM transformation is not to map the whole EAST-ADL metamodel to RubusMM, rather to map only the part of the EAST-ADL metamodel needed for synthesising RCM models along with the timing information needed for running high-precision timing analysis.

The DL2RCM model transformation has been implemented by means of *medini QVT*.[7] Medini QVT is an Eclipse Modeling Framework tool set for model to model transformations, which implements the OMG's QVT Relations standard [27]. For the sake of simplicity, algorithm 1 shows the DL2RCM transformation in pseudocode. [8] The MODEL2SYSTEM function is the entry point of the transformation. It is responsible for translating an EAST-ADL Model element into a hierarchy of RubusMM elements consisting of System, Node, Target and Mode elements (line 2). This step can be skipped when considering all EAST-ADL abstraction levels, since the RubusMM elements would be translated from the equivalent EAST-ADL elements. In our case, since we are considering just the EAST-ADL design level, this step is needed to build a correct hierarchy in the Rubus model, conforming to the RubusMM. One of the major difficulties in defining the DL2RCM transformation is that EAST-ADL implements the *type-prototype* pattern: a *DesignFunctionPrototype* element is considered to be a specific instance of the *DesignFunctionType* element which in turn might contain other prototypes and connectors realising its inner architecture (see Figure 16). This means that the inner architecture of a prototype is defined through its related type. Such a pattern, not leveraged in RubusMM, required

[6]The interested reader can find a detailed discussion on the assumptions and the constraints used for defining the DL2RCM transformation in [26].

[7]http://projects.ikv.de/qvt

[8]The interested reader can find the actual QVT code for the DL2RCM transformation at `http://www.mrtc.mdh.se/DL2RCM.qvt`.

Algorithm 1 DL2RCM Transformation

```
54: function DP2A(DesignFunctionPrototype dfp, Assembly a)
55:     Assembly as = createAssembly(dfp, a);
56:     for connector in dfp do
57:         c2c(connector, a)
58:     end for
59:     for part in dfp do
60:         if part is not elementary then
61:             Assembly as1 = DP2A(part, as);
62:         else
63:             Circuit c1 = DP2C(part, as);
64:         end if
65:     end for
66:     return as;
67: end function
68: function DP2C(DesignFunctionPrototype dfp, Assembly a)
69:     Circuit c = createCircuit(dfp, a);
70:     ConnectorTrig conTC = createControlFlowIn(dfp, c, a);
71:     ConnectorTrig conTS = createControlFlowOut(dfp, c, a);
72:     return c;
73: end function
74: function TC2TC(FunctionalDesignPrototype fdp, Mode mo)
75:     for tc in fdp do
76:         Event startTC = tc.scope.stimulus;
77:         Event endTC = tc.scope.response;
78:         ConnectorData conS = find(mo.assembly, startTC);
79:         ConnectorData conE = find(mo.assembly, endTC);
80:         if tc is AgeConstraint then
81:             DataAgeStart startA = createDataAgeStart(tc);
82:             DataAgeEnd endA = createDataAgeEnd(tc);
83:             assignPorts(startA, endA, conS, conE)
84:         else
85:         end if
86:         if tc is ReactionConstraint then
87:             DataReactionStart startR = createDataReactionStart(tc);
88:             DataReactionEnd endR = createDataReactionEnd(tc);
89:             assignPorts(startR, endR, conS, conE)
90:         else
91:         end if
92:     end for
93:     for part in fdp do
94:         TC2TC(part, mo.assembly)
95:     end for
96: end function
```

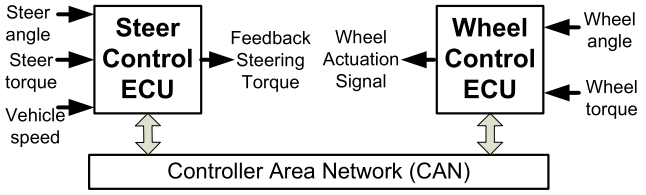

FIGURE 7. Architecture of the steer-by-wire system.

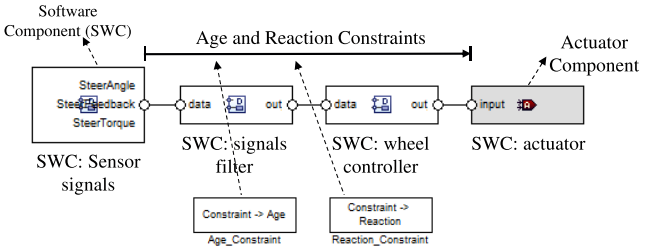

FIGURE 8. Software architecture of WC ECU modeled with EAST-ADL and TADL2.

additional effort in designing the transformation, as each DesignFunctionPrototype has to be checked against its type before to be transformed. These negligible low-level details are omitted from the pseudocode in Algorithm 1 for the sake of readability. For the same reason, in the pseudocode we make use of helper functions (e.g., CREATEHIERARCHY, CREATEASSEMBLY) which are responsible for the creation of the related elements and their inner architecture. The *FDP* function is responsible for translating an EAST-ADL *DesignFunctionPrototype* element into a RCM *Assembly* or *Circuit* element depending on whether its related *DesignFunctionType* is an elementary element (meaning that it does not contain any other *DesignFunctionProtype* element). In the case it is not an elementary element (line 14), all the contained *DesignFunctionProtype* elements are transformed too. This translation is performed in two steps. First, FDP calls the *C2C* function on all its *FunctionConnector* elements (lines 10-12), for the translation of the elements linked via connectors. Afterwards, the FDP function calls *DP2A* or *DP2C* on its spare *DesignFunctionProtype* elements; if they are elementary elements then they are transformed into circuits by the DP2C function (line 17), otherwise they are transformed into assemblies through the DP2A function (line 15). Figure 9 shows the Ecore model serialising the above mentioned architecture. The C2C function translates an EAST-ADL *FunctionConnector* element into a RCM *DataConnector* element. More precisely, for each *FunctionConnector* element, the C2C function creates a *DataConnector* element (line 26) together with the connected *Assembly*/*Circuit* elements by calling the functions DP2A (line 29) and DP2C (line 41), respectively. *Port* elements are created and connected accordingly (lines 33, 36, 45, 48). Control flow information is not explicitly modelled at EAST-ADL design level. Therefore, we assume that each SWC is triggered independently.

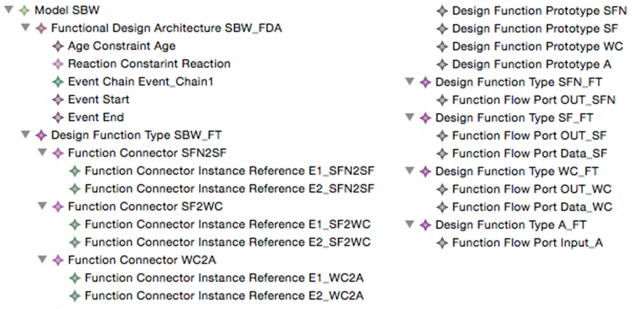

FIGURE 9. Serialized model of the EAST-ADL WC ECU architecture.

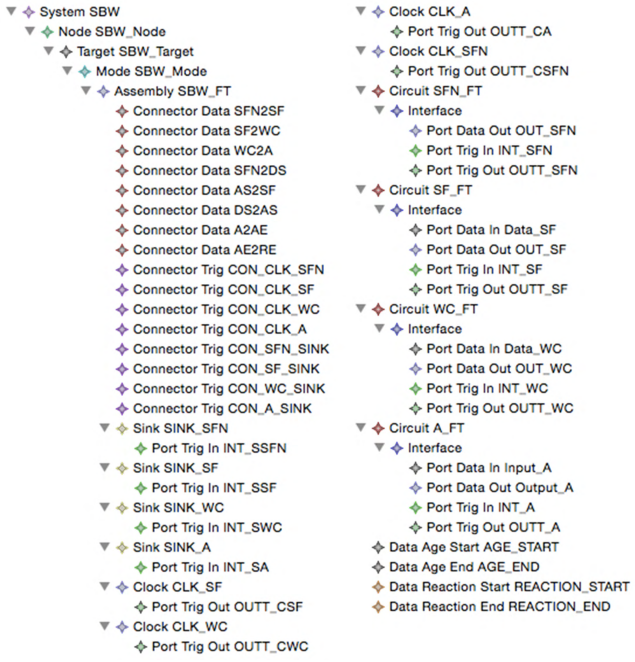

FIGURE 10. Serialized model of the RCM WC ECU architecture.

To this end, the C2C function generates the needed *Clock* (lines 32, 44) and *Sink* (lines 35, 47) elements together with the *ConnectorTrig* elements. With a logic similar to FDP, functions *DP2C* and *DP2A* translate an EAST-ADL DesignFunctionPrototype into RCM Circuit and RCM Assembly, respectively. The function *TC2TC* is responsible for translating the timing (age and reaction) constraints. Starting from the outer *DesignFunctionPrototype*, it iterates on all the specified timing constraints (line 78). For each of them, it uses the start and end events (*stimulus* and *response* in Figure 16) for searching, within the RCM model, the connector attached to the port and specified by the stimulus or response events (lines 79-82). After *DataAgeStart*, *DataReactionStart*, *DataAgeEnd* and*DataReactionEnd* elements are created (lines 84, 85, 90, 91), they are connected to the related data ports (lines 86, 92).

V. APPLICATION TO THE STEER-BY-WIRE SYSTEM

In order to show the applicability of the DL2RCM transformation, we exploit a portion of the Steer-By-Wire (SBW)

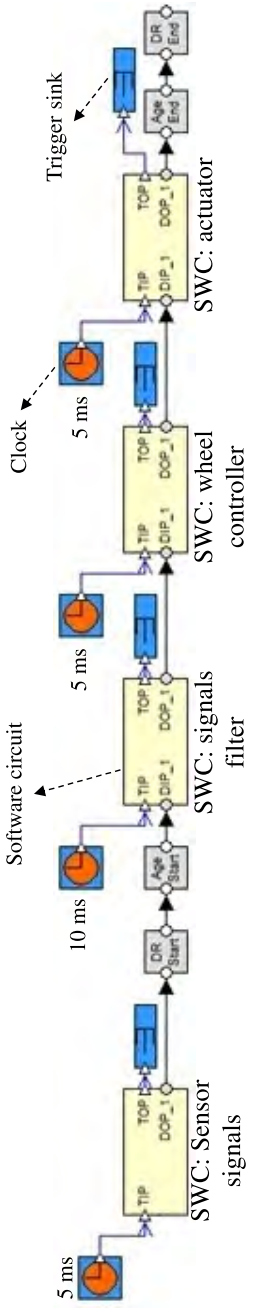

FIGURE 11. Translated software architecture of WC ECU in RCM.

system, which represents a vehicular feature that substitutes mechanical and hydraulic components with electronic components in the steering system of a vehicle.

A partial architecture of the SBW system is shown in Figure 7. There are two ECUs (rest of the ECUs are not shown for simplicity) that are connected to a single Controller Area Network (CAN) bus. The Steering Control (SC) ECU receives inputs from steering angle, steering torque and vehicle speed sensors. It also receives a CAN message from the Wheel Control (WC) ECU. It sends two CAN messages: one carries steer angle and torque signal, while the other carries feedback signals. Based on all the inputs, it calculates

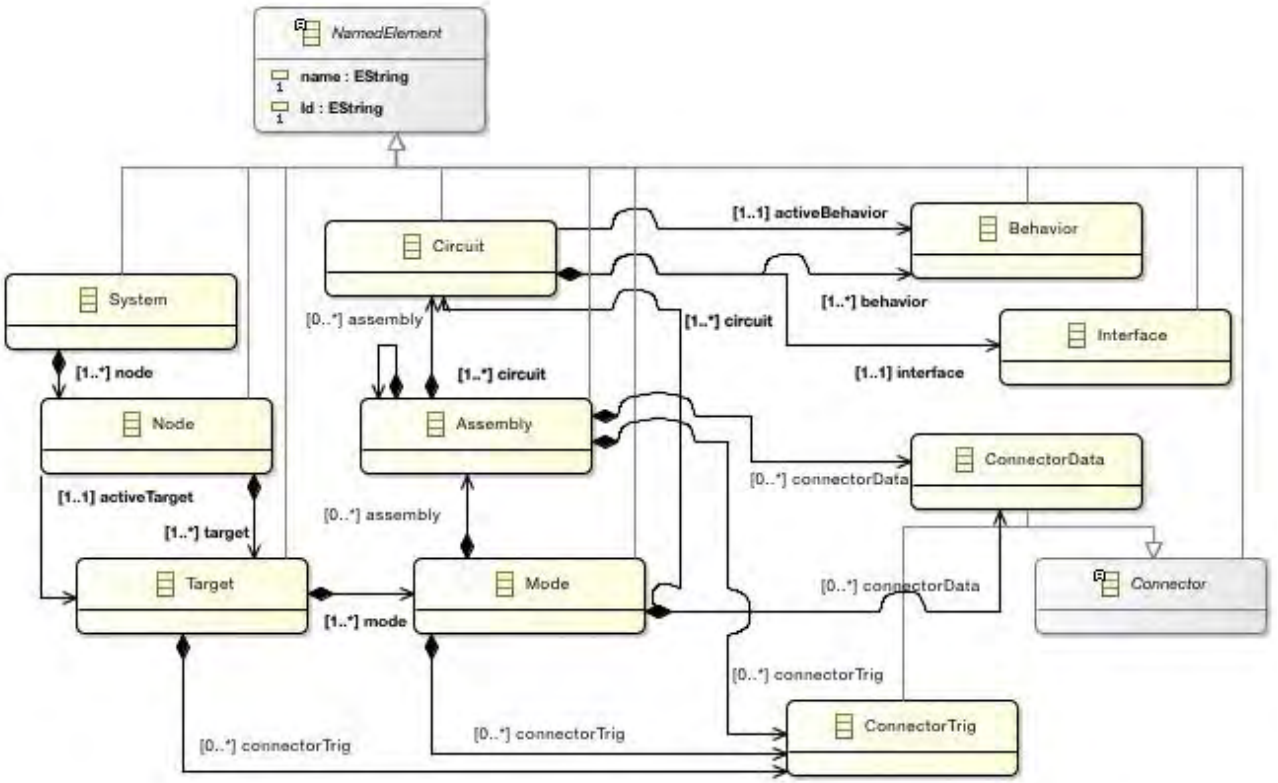

FIGURE 12. Fragment of the RCM metamodel representing the backbone elements.

the feedback steering torque and sends it to the feedback torque actuator which is responsible for producing the turning effect of the steering. Similarly, the WC ECU receives inputs from wheel angle and torque sensors. Depending upon these signals and CAN messages received from the SC ECU, it calculates the wheel torque and produces actuation signals for the wheel actuators. It also sends one CAN message carrying wheel torque signal. For the sake of simplicity and intuitive presentation of the transformation, the simplified internal software architecture of WC ECU is modelled with EAST-ADL using EAST-ADL Rubus Designer[9] as shown in Figure 8. There are four EAST-ADL Software Components (EastSWCs) in the simplified software architecture. We specify two timing constraints, namely age and reaction using TADL2. These constraints put a restriction of 20 ms on the time between the acquisition of sensor signals at the WC ECU and the production of wheel actuation signals by the actuator EastSWC. These constraints are internally referenced to the components on which they are specified. For convenience, the start and end points for these constraints are identified using the solid-line arrow.

Applying the DL2RCM transformation presented in Section IV, the Ecore model in Figure 10 is obtained. Without going into the details of the transformation process, it can be noted how the RCM elements were translated from the related EAST-ADL elements. For instance, the RCM SWC SFN_FT has been translated from the EAST-ADL DesignFunction-Type SFN_FT by means of the C2C function. The same applies to all the RCM elements. A representation, given in Rubus Designer concrete syntax, of the model showed in Figure 10, is presented in Figure 11. The specified TADL2 timing constraints (i.e., Age and Reaction) in Figure 8 are also translated to RCM timing constraints shown by ''Age Start'', ''Age End'', ''DR Start'', and ''DR End'' objects in Figure 11. We make three assumptions to support the analysis of the translated software architecture in RubusMM. The first assumption concerns the priority of the four tasks (run-time entities) that correspond to the four EastSWCs in Figure 11. EAST-ADL does not support specification of priorities on the software components. In order to consider the worst-case scenario, where each of the four tasks is assumed to be interfered by the rest of the tasks, we assume that the priorities of the four tasks are equal. Secondly, we assume that the four tasks are the highest priority tasks and these tasks do not experience any blocking from the other tasks in the WC ECU. This assumption is needed to support the analysis since we have considered the reduced software architecture in Figure 10. The third assumption concerns the WCETs of the components shown in Figure 11. The WCETs are selected, based on the experience from similar cases studies, between the range 60 μs - 2000 μs. The analysis engines calculate the age and reaction delays for only those component chains

[9]http://www.arcticus-systems.com

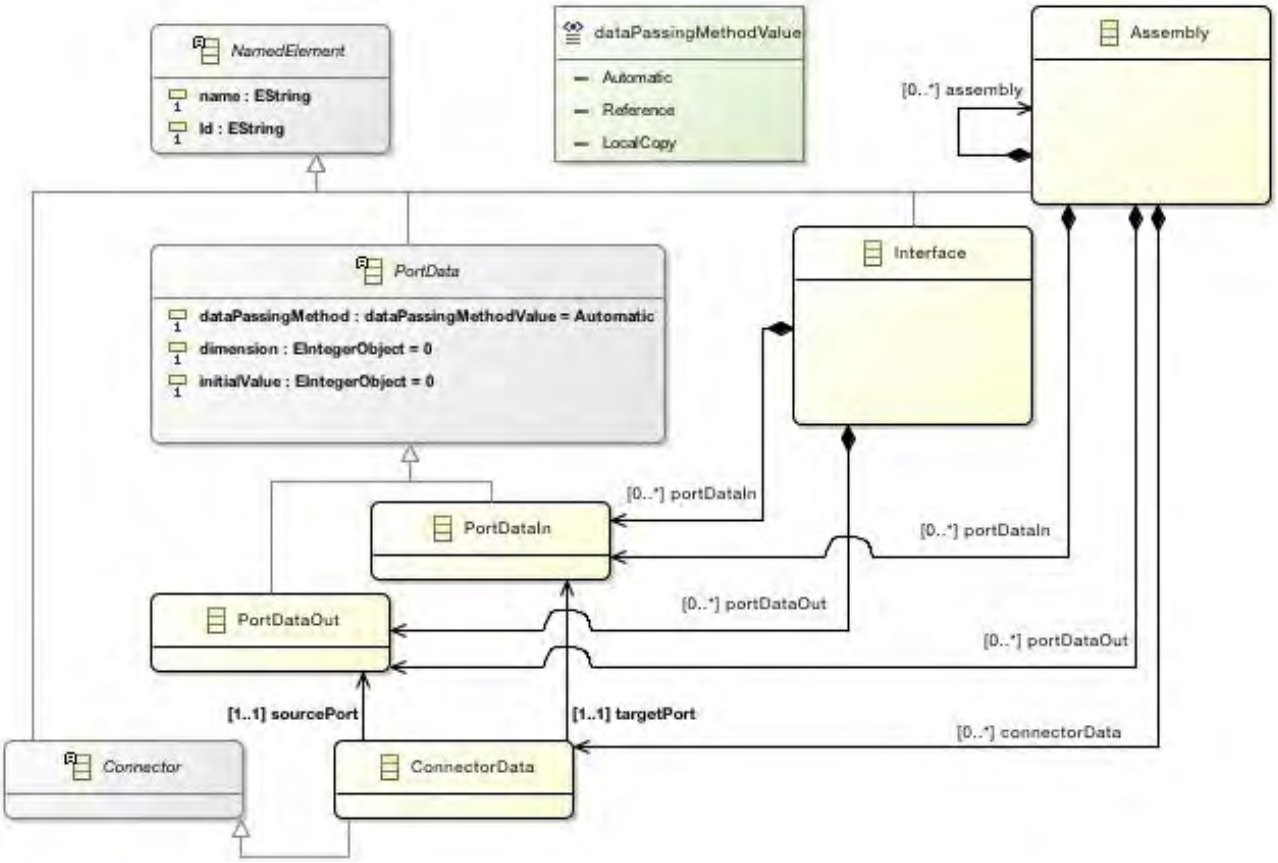

FIGURE 13. Fragment of the RCM metamodel representing the data flow elements.

(represented by task chains at runtime) on which the timing constraints are specified (there is only one component chain in Figure 11 on which these delays are specified). The calculated age and reaction delays are 5360 μs and 15360 μs respectively. A comparison between the specified constraints and calculated delays shows that the system satisfies the specified timing constraints.

VI. VALIDATION AND DISCUSSION

The mismatch between the structural and semantic assumptions of the plethora of different modelling languages currently used in the software development hampers interoperability. In the automotive domain, an example of this phenomenon is the semantic gap between modelling languages used for functional modelling (e.g., EAST-ADL) and those used for implementation modelling (e.g., RCM). One way to ensure interoperability is to employ MDE for defining automatic translations of the models specified using, e.g., EAST-ADL and RCM. In this respect defining appropriate metamodels is a fundamental step towards enabling the implementation of MDE techniques. As a consequence, the RCM metamodel has been developed with two aspects in mind: *backward compatibility* and *extensibility*. The first aspect has been addressed by reverse engineering the internal representation of RCM into the Rubus Integrated Component Model Development Environment (RUBUS ICE). Redundancies, due to the lower level of abstraction, have been polished and model traversals improved. These activities resulted in the addition of 6 elements and the refinement of 5 hierarchy elements. Please note that the refinement activities done within the RubusMM definition do not affect its expressiveness which has been discussed and validated already in [7]. The correctness of the metamodel illustrated in this paper has been checked against several existing industrial system designs, e.g., modelling of i) Autonomous Cruise Control System that consists of 4 nodes (ECUs), 17 assemblies and 36 SWCs [5], ii) Intelligent Parking Assist (IPA) System that consists of 2 nodes and 42 SWCs [28], simplified IPA system consisting of 2 nodes and 7 SWCs [29] and iii) simplified Steer-by-wire System consisting of 1 node and 6 SWCs [30]. Extensibility targets the general trend of incrementally adopting higher abstraction level approaches to deal with the development of industrial systems (see also the discussion that follows in the remainder of this section). In our specific cases, it requires RUBUS ICE to be open enough to be integrated in a tool chain.

The proposed metamodel-based solution supports tool integration contexts by permitting the definition of model

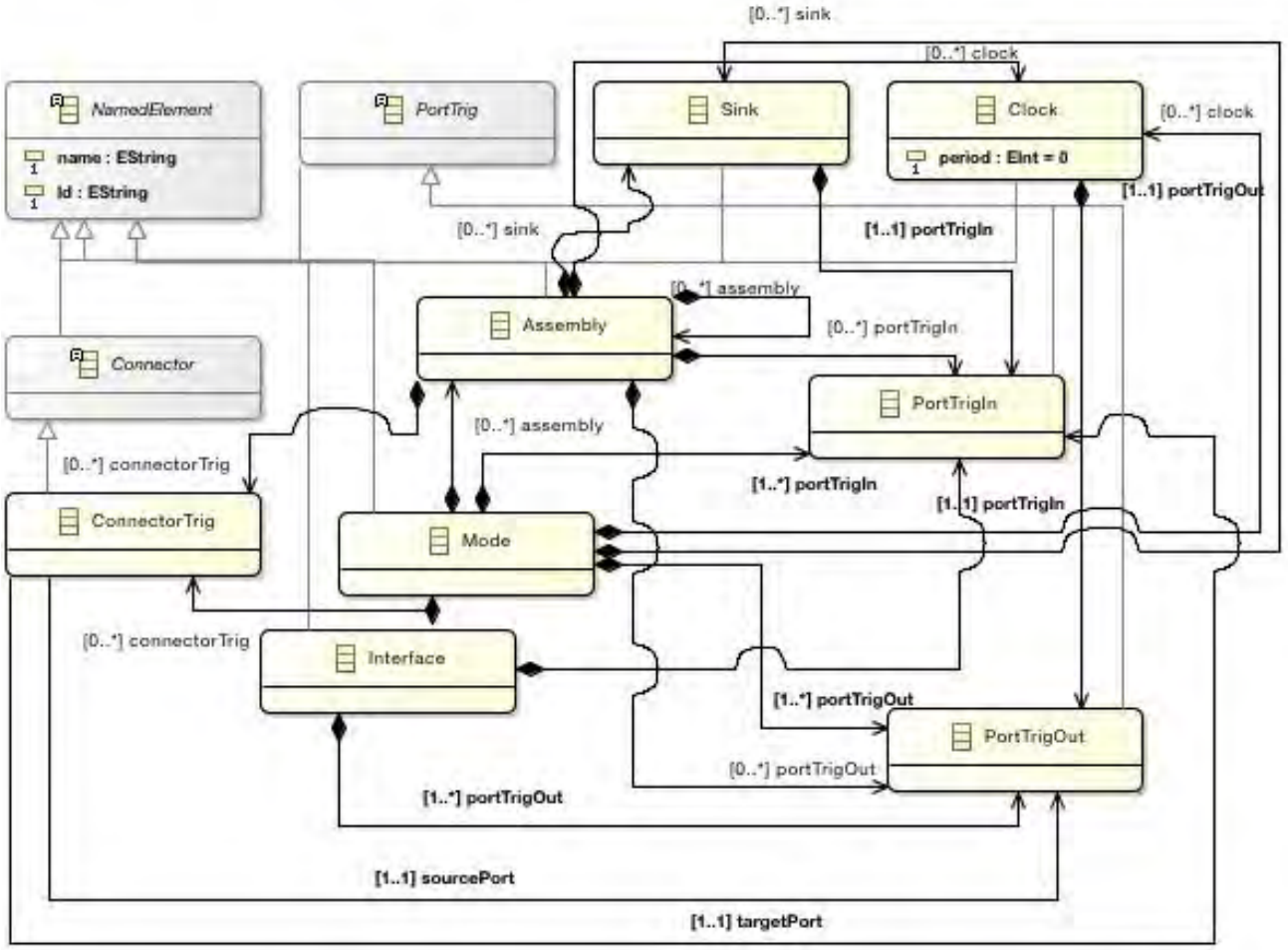

FIGURE 14. Fragment of the RCM metamodel representing the control flow elements.

transformations acting as import/export utilities from a tool to another. The transformation from EAST-ADL to RubusMM and its application illustrated in Section IV and V, respectively, are a practical demonstration of the tool integration potentials disclosed by the adoption of a model-driven approach. Writing and testing the tool integration transformation is a one time effort; then the translation can be used for all the models defined in the tools, as long as the metamodels are not modified. The correctness of the model transformation has been tested upon the above mentioned industrial systems designs. Moreover, we have used synthetic models for verifying possible unexpected behaviours of the transformation. To this end, we created class of models containing 13, 40, 187 and 1000 elements, respectively. These models were also used for validating the transformation performance and scalability. In particular, the transformation has been run 10 times for each class of models. Table 2 reports, for each class of model, the average of the ten execution times.[10] Table 2 also shows the size of the target models generated by the DL2RCM transformation. It is worth mentioning that the growth in size of the generated target models is not linear to the growth in size of the source models, when source models have dense connections. In fact, each FunctionConnector (together with its own inner architecture) in the source models would be translated in a DataConnector and TrigConnector.

TABLE 2. Execution times of the DL2RCM transformation.

Number of Elements in EAST-ADL Model	*Number of Elements in RubusMM Model*	*Transformation Time (ms)*
13	10	63
40	50	100.1
187	258	390.9
1000	1856	12195.3

However, with source models with less dense connections, it is possible to observe a reduction on the size of the generated target models. This is due to the type-prototype pattern used in EAST-ADL, but absent in RubusMM. With respect to the maintainability aspect, building-up the development environment on the RCM metamodel allows to decou-

[10]The interested reader can find here `http://www.mrtc.mdh.se/DL2RCMExecutionTime.pdf` the extended report containing all the data from all the transformation executions.

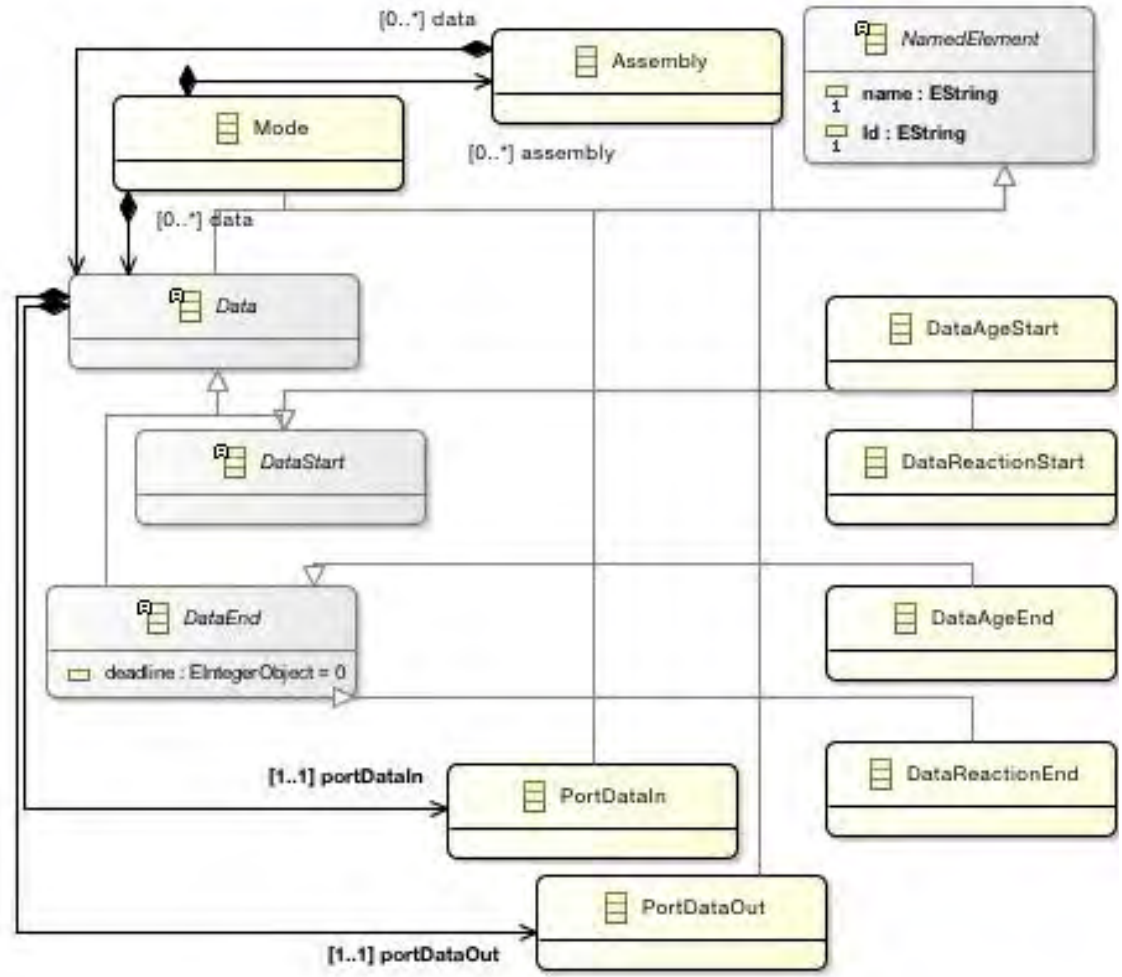

FIGURE 15. Fragment of the RCM metamodel representing the timing constraints and properties for different types of delay in event chains.

ple modelling concepts from their rendering and from the automated features provided as part of the tool. This means that extensions/refinements of RCM cause modifications of the current metamodel, which in turn trigger co-evolutions of interconnected artefacts [31]. Despite managing metamodel evolutions is not always straightforward, having an explicit link between RCM changes and metamodel manipulations allows to perform an impact analysis of the refinements and to precisely locate where changes will affect existing artefacts. Notably, especially in industrial contexts the need for local customisations of tools requiring *ad hoc* adaptations can arise. On the one hand, operating at a higher level of abstraction allows to show/hide modelling elements, increase/reduce the number of modelling views, and so on. On the other hand, the need for metamodel modifications limits the dangerous practice of hardcoding customisations directly on the implementation code of the modelling environment, which hinders its maintainability in the long run. From a broader perspective, introducing approaches leveraging higher level of abstraction for the development of complex systems is an indisputable trend in modern software engineering practice. In this respect, industry is very often facing the issue of integrating new task-specific tools with legacy systems and development environments. In particular, if the constellation of adopted tools is not integrated in a seamless chain, manual effort is required to close the gap between tools and perform needed translations. Even if feasible, this practice can become time-consuming and error-prone in the long run, especially when the size of the system grows and there are semantics aspects involved in the translations. Model transformations automate the integration process between tools by translation means and can provide explicit traceability of the translations. Traces not only allow to explicitly represent the correspondences between one tool and another, but they also enable the propagation of information from one domain-specific perspective to another. Notably, in the example presented in Section V the forward transformation allows to get an Rubus model from EAST-ADL, and carries the rationale underlying the mapping across these two languages. Moreover, the trace links created during the transformation process

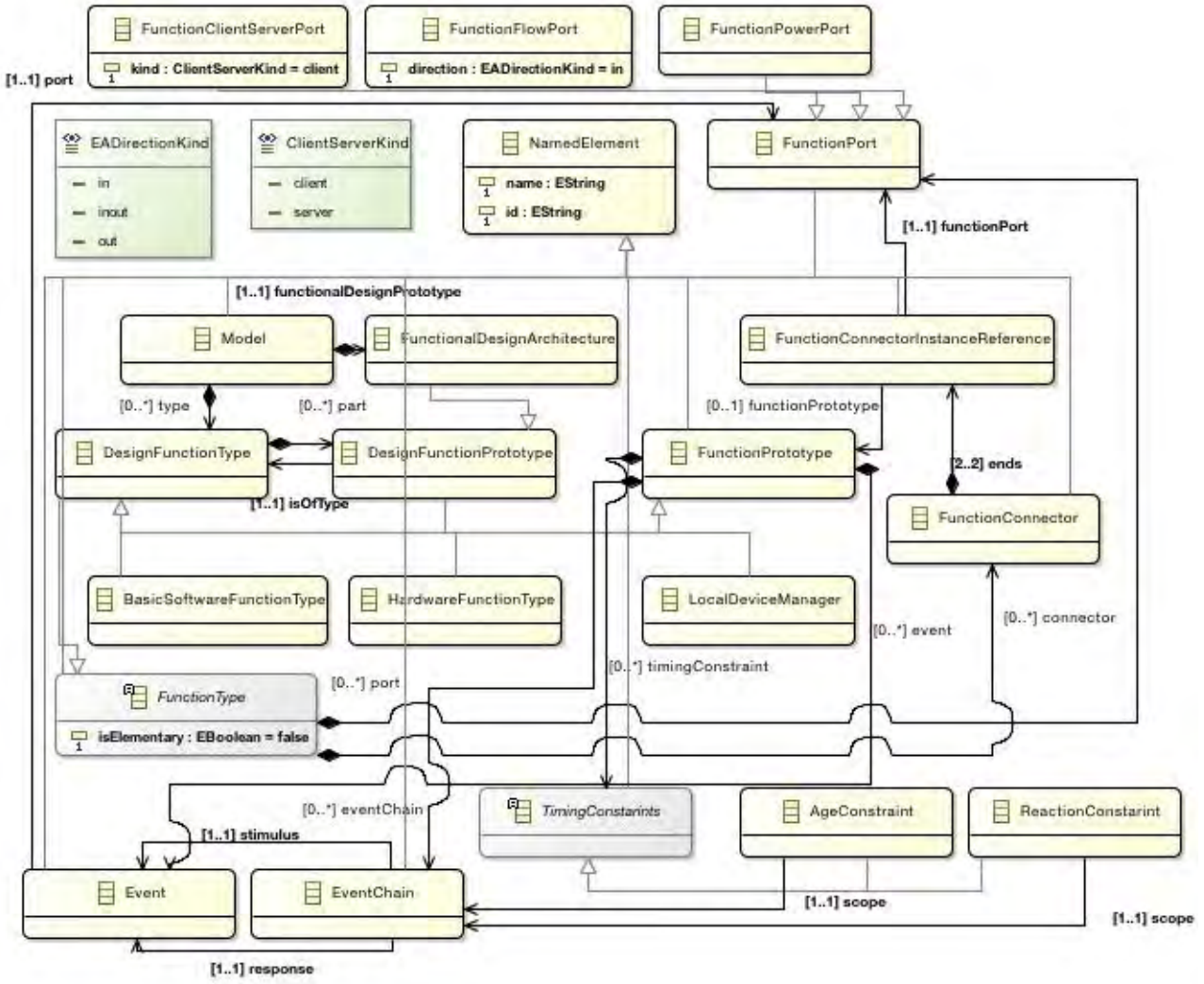

FIGURE 16. Fragment of the EAST-ADL metamodel for Function Modelling at the design level.

allow, for example, to map timing analysis results back to EAST-ADL models.

VII. CONCLUSIONS AND FUTURE WORK

In the last twenty years, 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. Intertwining of MDE and CBSE has been proven to be effective towards this goal.

In this work, by exploiting the interplay between MDE and CBSE, we took initial steps towards the realization of the aforesaid seamless development chain. In details, we i) motivated the usage of RCM in the vehicular domain, by highlighting its unique features against other CMs, ii) formalized a metamodel based on RCM comprising the concepts able to represent both the software architecture and the related timing constraints,, iii) presented a model-to-model transformation between EAST-ADL Design level and RCM and iv) discussed the application of our solution to an automotive industrial application. The formalization of the metamodel serves as basis for embracing the MDE vision as well as for restoring the separation of concerns that had been lost during the evolution of the RCM. Due to space limitations, we did not discuss the complete RCM timing package, but we rather focused on the elements representing timing constraints, information and analyses practically used within the industrial automotive domain. The DL2RCM transformation outlines the potential benefits gained in having a proper metamodel for RCM, in terms of compliance with the EASTADL based methodology.

As future work, we plan to minimize the assumptions needed in performing the transformation, by using model transformation languages able to fully and practically support non-bijective model transformations. Additionally, we will consider the possibility of using these non-bijective model transformations for design-space exploration. Finally we will, 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.

ACKNOWLEDGEMENT

The authors thank their industrial partners Arcticus Systems AB and Volvo CE, Sweden.

REFERENCES

[1] R. N. Charette, "This car runs on code," *IEEE Spectr.*, vol. 46, no. 3, p. 3, 2009.
[2] F. Liu, A. Narayanan, and Q. Bai, "Real-time systems," Tech. Rep., 2000.
[3] T. A. Henzinger and J. Sifakis, "The embedded systems design challenge," in *Proc. 14th Int. Symp. Formal Methods (FM)*, 2006, pp. 1–15.
[4] S. Kent, "Model driven engineering," in *Proc. Int. Conf. Integr. Formal Methods*, 2002, pp. 286–298.
[5] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "Support for end-to-end response-time and delay analysis in the industrial tool suite: Issues, experiences and a case study," *Comput. Sci. Inf. Syst.*, vol. 10, no. 1, pp. 453–482, 2013.
[6] (Oct. 2013). *AUTOSAR Techincal Overview, Release 4.1, Rev. 2, Ver. 1.1.0., The AUTOSAR Consortium*. [Online]. Available: http://autosar.org
[7] K. Hänninen, J. Mäki-Turja, M. Nolin, M. Lindberg, J. Lundback, and K.-L. Lundback, "The Rubus component model for resource constrained real-time systems," in *Proc. 3rd IEEE Int. Symp. Ind. Embedded Syst.*, Jun. 2008, pp. 177–183.
[8] (2010). *EAST-ADL Domain Model Specification, Deliverable D4.1.1*. [Online]. Available: http://www.atesst.org/home/liblocal/docs/ATESST2_D4.1.1_EAST-ADL2-Specification_2010-06-02.pdf
[9] P. Cuenot *et al.*, "11 The EAST-ADL architecture description language for automotive embedded software," in *Model-Based Engineering of Embedded Real-Time Systems*. Springer, 2011, pp. 297–307.
[10] G. Liebel, N. Marko, M. Tichy, A. Leitner, and J. Hansson, "Assessing the state-of-practice of model-based engineering in the embedded systems domain," in in *Proc. Int. Conf. Model Driven Eng. Lang. Syst.*, 2014, pp. 166–182.
[11] J. Bézivin and O. Gerbé, "Towards a precise definition of the OMG/MDA framework," in *Proc. 16th IEEE Int. Conf. Autom. Softw. Eng.*, Nov. 2001, pp. 273–280.
[12] S. Sendall, W. Kozaczynski, "Model transformation: The heart and soul of model-driven software development," *IEEE Softw.*, vol. 20, no. 5, pp. 42–45, Sep. 2003.
[13] (Sep. 2016). *Specification of the Virtual Functional Bus, Release 3.0, Rev. 7, Ver. 1.1.0., The Autosar Consortium*. [Online]. Available: https://www.autosar.org/fileadmin/files/releases/3-0/main/standard/AUTOSAR_SWS_VFB.pdf
[14] Methodology, TIMMO, "Version 2," *TIMMO (TIMing MOdel), Deliverable*, vol. 7, 2009. [Online]. Available: https://itea3.org/project/timmo.html
[15] *TIMMO-2-USE*, accessed on Sep. 2016. [Online]. Available: https://itea3.org/project/timmo-2-use.html
[16] *TADL: Timing Augmented Description Language, Version 2, Deliverable 6, The TIMMO Consortium*, Oct. 2009.
[17] OMG Group. (2010). *The UML Profile for MARTE: Modeling and Analysis of Real-Time and Embedded Systems*. [Online]. Available: http://www.omgmarte.org/
[18] *Timing Augmented Description Language (TADL2) Syntax, Semantics, Metamodel Ver. 2, Deliverable 11*, Aug. 2012.
[19] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "Communications-oriented development of component-based vehicular distributed real-time embedded systems," *J. Syst. Archit.*, vol. 60, no. 2, pp. 207–220, 2014.
[20] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocess. Microprogram.*, vol. 40, pp. 117–134, Apr. 1994.
[21] F. Stappert, J. Jonsson, J. Mottok, and R. Johansson, "A design framework for end-to-end timing constrained automotive applications," in *Proc. Embedded Real-Time Softw. Syst. (ERTS)*, 2010.
[22] N. Feiertag, K. Richter, J. Nordlander, and J. Jonsson, "A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics," in *Proc. Workshop Compositional Theory Technol. Real-Time Embedded Syst. (CRTS)*, 2008, pp. 1–8.
[23] A. C. Rajeev, S. Mohalik, M. G. Dixit, D. B. Chokshi, and S. Ramesh, "Schedulability and end-to-end latency in distributed ECU networks: Formal modeling and precise estimation," in *Proc. 10th ACM Int. Conf. Embedded Softw. (EMSOFT)*, 2010, pp. 129–138.
[24] A. Bucaioni, A. Cicchetti, and M. Sjödin, "Towards a metamodel for the rubus component model," in *Proc. 1st Int. Workshop Model-Driven Eng. Component-Based Softw. Syst. (ModComp)*, 2014, pp. 46–56.
[25] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "Translating timing constraints during vehicular distributed embedded systems development," in *Proc. 1st Int. Workshop Model-Driven Eng. Component-Based Softw. Syst.*, 2014, pp. 57–66. [Online]. Available: http://www.es.mdh.se/publications/3656-
[26] A. Bucaioni, S. Mubeen, A. Cicchetti, and M. Sjödin, "Exploring timing model extractions at EAST-ADL design-level using model transformations," in *Proc. Int. Conf. Inf. Technol., New Generat. (ITNG)*, 2015, pp. 595–600.
[27] OMG. (Sep. 2016). *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, Ver. 1.1*. [Online]. Available: http://www.omg.org/spec/QVT/1.1/PDF/
[28] A. Bucaioni, S. Mubeen, J. Lundbäck, K.-L. Lundbäck, J. Mäki-Turja, and M. Sjödin, "From modeling to deployment of component-based vehicular distributed real-time systems," in *Proc. Int. Conf. Inf. Technol., New Generat. (ITNG)*, 2014, pp. 649–654.
[29] A. Bucaioni, A. Cicchetti, F. Ciccozzi, S. Mubeen, M. Sjödin, and A. Pierantonio, "Handling uncertainty in automatically generated implementation models in the automotive domain," in *Proc. 42nd Euromicro Conf. Ser. Softw. Eng. Adv. Appl.*, 2016, pp. 173–180. [Online]. Available: http://www.es.mdh.se/publications/4362-
[30] A. Bucaioni, A. Cicchetti, F. Ciccozzi, R. Eramo, S. Mubeen, and M. Sjödin, "Anticipating implementation-level timing analysis for driving design-level decisions in EAST-ADL," in *Proc. Int. Workshop Modelling Automotive Soft. Eng.*, 2015, pp. 63–72. [Online]. Available: http://www.es.mdh.se/publications/4022-
[31] D. Di Ruscio, L. Iovino, and A. Pierantonio, "What is needed for managing co-evolution in MDE?" in *Proc. 2nd Int. Workshop Model Comparison Pract.*, 2011, pp. 30–38.

ALESSIO BUCAIONI (M'13) received the M.Sc. degree in software engineering from Mälardalen University, Sweden, and the M.Sc. degree (Hons.) in software engineering from the University of L'Aquila, within the Global Software Engineering European Master, an international double degree master program. He is currently pursuing the Ph.D. degree with the School of Innovation, Design and Engineering, Mälardalen University. He also works as a Software Engineering for Arcticus Systems AB, Sweden. His research interests include model-driven engineering for embedded systems, software engineering and architecture, and software development, focusing on how to improve the model-based development of vehicular embedded systems.

ANTONIO CICCHETTI (SM'09) received the Ph.D. degree in computer science from the Computer Science Department, University of LAquila, in 2008. He is currently an Associate Professor with Mälardalen University, Västerås, Sweden. His current research interests include MDE, model versioning, metamodeling, model transformations and weaving, generative techniques in Web engineering, methodologies for Web development, model versioning, as model repositories, model co-evolution and synchronization, and the application of MDE techniques to the component-based development field, with respect to system modelling, generation of code, and verification and validation activities.

FEDERICO CICCOZZI (SM'09) is currently a Senior Lecturer with Model- based engineering for embedded systems and Industrial software engineering groups, Mälardalen University. His research focuses on several aspects of model-driven engineering, component-based software engineering for the development of embedded systems based on domain- specific modelling languages (DSMLs), both UML- and EMF-based, the definition of DSMLs, automatic model manipulations through transformations, system properties preservation multi-paradigm modelling, model versioning, (co)evolution and synchronization, and the application of MDE and CBSE techniques to mobile multi-robot systems.

SAAD MUBEEN (SM'06) received the Ph.D. degree in computer science and engineering from Mälardalen University, Sweden, in 2014. He was a Software Engineer and a Consultant with Arcticus Systems and Volvo Construction Equipment, Sweden, respectively. He is currently a Senior Lecturer/Assistant Professor with the School of Innovation, Design and Engineering, Mälardalen University. He has co-authored over 85 publications in international peer-reviewed journals, conferences, workshops, and book chapters. His research interests include the model-based development of vehicular embedded systems with a focus on timing models extraction, end-to-end timing analysis, and multicore platforms.

MIKAEL SJÖDIN received the Ph.D. degree in computer systems from Uppsala University, Sweden, in 2000.is currently a Professor with Mälardalen University, Västerås, Sweden. He was with Newline Information, Melody Interactive Solutions, and CC Systems. Since then, he has been involved in both academia and in industry with embedded systems, real-time systems, and embedded communications. In 2006, he joined the MRTC faculty, Mälardalen University, as a Full Professor with specialty in real-time systems and vehicular software-systems. His research focuses on new methods to construct software for embedded control systems in the vehicular and telecom industry, and to find methods that will make software development cheaper, faster and yield software with higher quality, the analysis of real-time systems, where the goal is to find theoretical models for real-time systems that will allow their timing behavior and memory consumption to be calculated.