MoVES: a Model-driven methodology for Vehicular Embedded Systems

Alessio Bucaioni*†, Lorenzo Addazi*, Antonio Cicchetti*, Federico Ciccozzi*,
Romina Eramo‡, Saad Mubeen*†, Mikael Sjödin*

* Mälardalen University, Västerås, Sweden
† Arcticus Systems AB, Järfälla, Sweden
‡ University of L'Aquila, L'Aquila, Italy

* {alessio.bucaioni, lorenzo.addazi, antonio.cicchetti, federico.ciccozzi, saad.mubeen, mikael.sjodin}@mdh.se
† {alessio.bucaioni, saad.mubeen}@arcticus-systems.com
‡ {romina.eramo}@univaq.it

Abstract—This paper introduces a novel model-driven methodology for the software development of real-time distributed vehicular embedded systems on single- and multi-core platforms. The proposed methodology discloses the opportunity of improving the cost-efficiency of the development process by providing automated support to identify viable design solutions with respect to selected non-functional requirements. To this end, it leverages the interplay of modelling languages for the vehicular domain whose integration is achieved by a suite of model transformations. An instantiation of the methodology is discussed for timing requirements, which are among the most critical ones for vehicular systems. To support the design of temporally correct systems, a cooperation between EAST-ADL and the Rubus Component Model is opportunely built-up by means of model transformations, enabling timing-aware design and model-based timing analysis of the system. The applicability of the methodology is demonstrated as proof of concepts on industrial use cases performed in cooperation with our industrial partners.

Keywords—*Model-driven development; vehicular embedded systems; EAST-ADL; component model; model transformations.*

I. INTRODUCTION

As vehicles transitioned from being mechanical-intensive to software-intensive systems [1], a cost-effective software development became paramount in the vehicular domain. Researchers and practitioners agreed that *abstraction* and *automation*, the founding pillars of Model-Driven Engineering (MDE), could be game changers towards the achievement of a cost-effective software development process as they contribute to shorten the development time and employ expensive resources more efficiently [2]. To this end, several domain-specific modelling languages were introduced both for designing the vehicular software and for representing its non functional properties such as timing. Among other languages, the Electronics Architecture and Software Technology Architecture Description Language (EAST-ADL) has been developed by the automotive industry to support modelling of vehicular functions and both their software realisation and desired non functional properties [3]. Within EAST-ADL, the system modelling is performed at four different levels of abstraction which are vehicle-, analysis-, design- and implementation-level (from highest to lowest abstraction level). The requirements on the vehicle functionality of the system are captured at the vehicle

level. At the analysis level, the system is defined in terms of abstract functional architecture with a provision for high-level analyses. Typically, the vehicular software is modelled at the design level by means of function, hardware and allocation models. At the implementation level, the design models are enriched with detailed execution information on e.g., timing (worst-case execution time, etc.). The implementation models are defined by means of other languages, such as the AUTomotive Open System ARchitecture (AUTOSAR) [4] or the Rubus Component Model (RCM) [5]. Often, implementation models are used as the base for code synthesis. However, support to models integration (e.g., among EAST-ADL and RCM models) in the development of vehicular embedded systems is still scarce and the translation from design- to implementation-level is mainly performed manually. Too often, this lack of automation defers the verification of non functional properties to the last phases of the development process. Empirical research shows that modifications during these phases can be 40 times more expensive than the same modifications done during the design of the software and can introduce inconsistencies if they are not properly back propagated [6].

In this context, our hypothesis is that providing automation for model integration would enable early verification of non functional requirements (e.g. timing requirements) during the design of vehicular embedded systems thus improving the cost-efficiency of their development. In fact, early verification of non functional requirements would limit the need for expensive modifications on the almost ready-to-deliver software, and automation would reduce the overall development time as well as enhance the use of expensive (engineering) resources.

In this paper, we propose a solution for early verification of non functional requirements by introducing MoVES, a model-driven methodology for the development of distributed vehicular real-time embedded systems on single- and multi-core platforms. Considering the importance of timing in the development of vehicular real-time systems, as acknowledged by several international projects and industrial initiatives (TIMMO-2-USE[1] and AUTOSAR[2].), the proposed methodology is instantiated to support the development and architectural exploration of system-designs with temporal awareness.

---

[1] https://itea3.org/project/timmo-2-use.html
[2] https://www.autosar.org

MoVES leverages the interplay of EAST-ADL and RCM for expressing functional and implementation models, respectively. Moreover, it features a fully automated mechanism defined in terms of six different model transformations that describe precise relationships between EAST-ADL and RCM. In particular, starting from the EAST-ADL function and hardware models, model transformations generate a set of RCM models. Model transformations automatically generate allocation information on the RCM models from the EAST-ADL allocation model, too. As there might be multiple implementation models for the same design, a source EAST-ADL model cannot be univocally translated into a single target RCM model [7]. Currently, most approaches only consider one particular model out of the many possible alternatives [8]. In this work, we leverage the properties of a constraint-based transformation language, the Janus Transformation Language [9] (JTL), to automatically derive all the possible RCM models entailing meaningful and unique timing and allocation configurations. Timing analysis is run on the generated RCM models. Eventually, model transformations propagate the generated RCM models and the related timing verification results to the design level for enabling timing-aware design decisions. It is important to note that, the process of generating and analysing RCM models is transparent to the engineer and can be guided by means of logic constraints. Moreover, the engineer does not have to manually define or investigate RCM models, but rather select the preferred RCM models from the set of the automatically generated ones. We validated MoVES through a set proof of concepts conducted in tight cooperation with our industrial partners in the automotive domain. These showed promising results in terms of i) applicability of the methodology and ii) reduction of late modifications at implementation level. The main scientific contributions brought by MoVES are:

- a mechanism for the automatic translation of design models into implementation models,
- a mechanism for the automatic allocation of software to hardware, and
- a mechanism for the back-propagation of the verification results and related implementation models to design models.

The rest of the paper is organised as follows. Section II sets the background for this research work along with its contributions and relations with authors' previous work. Section III describes the methodology and its constituents. Section V describes the application of the methodology on a running example mimicking industrial settings. Section VI discusses strengths and weakness of the proposed methodology. Section VII describes related approaches documented in the literature and compares them to our solution. Finally, Section VIII concludes the paper.

II. BACKGROUND

MDE is a discipline which aims at improving software development by employing abstraction and automation by using models, metamodels and model transformations [2]. Metamodels formalise the requirements, the structure and the behaviour of software systems within a particular domain. Models allow to design software systems declaratively using the elements and the concepts formalised by the metamodels, thus using constructs pertaining to the problem domain rather than constructs pertaining to the underlying technology. Model transformations are automatic means for analysing models and for synthesising new artefacts (models, source code, etc.) from a set of source models [10]. In the automotive domain, as vehicle transitioned from being mechanical- to software-intensive systems [11], MDE has gained industrial recognition as demonstrated by several international initiatives and projects, such as EAST-ADL [3], RCM [5] and AUTOSAR [4].

In the followings, we describe the background of this research work and its contribution in terms of the modelling languages and the model-based timing analysis leveraged for the definition of MoVES. In particular, in Section II-A and in Section II-B we introduce and describe the main elements of the EAST-ADL and RCM languages, respectively. In Section II-C we discuss the leveraged timing analysis, while in Section II-D we detail the contributions presented in this paper and put them in relation to the authors' previous work.

A. EAST-ADL

EAST-ADL is a modelling language which captures the essentials of vehicular Electrical and Electronic (E/E) systems concerning their documentation, design, analysis and synthesis. EAST-ADL is specified through ten different packages, each of which addresses different aspects of vehicular E/E system and their development. In the proposed instantiation of the methodology, we leverage specific concepts from the *structure*, *requirements* and *timing* packages[3].

The structure package serves for the specification of the software architecture in terms of basic elements and interactions among them. In order to ensure separation of concerns through the development process, the structure package makes use of four abstraction levels, which are *vehicle*, *analysis*, *design* and *implementation*. However, such a separation is only conceptual and some modelling elements can span over several abstraction levels. MoVES connects to the design level and more specifically to the *FunctionalDesignArchitecture*, *HardwareDesignArchitecture* and *Allocation* concepts.

Within EAST-ADL, a FunctionalDesignArchitecture describes how software functions interact. At this level, software functions are represented by means of *DesignFunctionPrototype* elements linked by *FunctionConnector* elements. A DesignFunctionPrototype is typed to a *DesignFunctionType* element which specifies its interface, in terms of *FunctionPort* elements, and its inner architecture, in terms of additional DesignFuntionPrototype elements. Within EAST-ADL, a HardwareDesignArchitecture describes the physical architecture of the vehicular embedded system. The basic modelling entity is the *HardwareComponentPrototype* which is typed to a *HardwareComponentType*. The *HardwarePortConnector* elements model the communication between two or more HardwareComponentPrototype elements by connecting the related *HardwarePort* elements. An EAST-ADL Allocation model consists of a set of *FunctionAllocation* elements binding *AllocatableElement* to *AllocationTarget* elements. AllocatableElement is an abstract superclass which specifies the elements that can be allocated. DesignFunctionPrototype and FunctionConnector are defined as extensions of the AllocatableElement

---

[3]Please note that the complete explanation of EAST-ADL is outside the scope of this work. The interested reader can refer to [3].
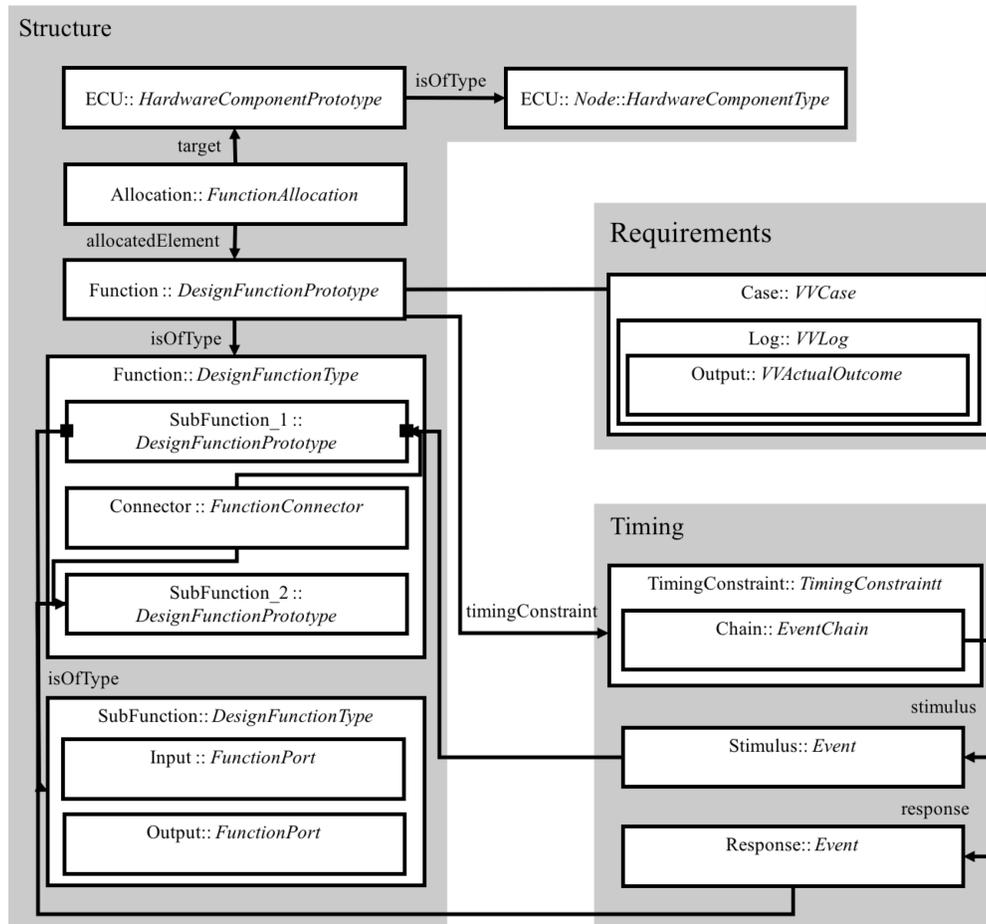
Fig. 1: Simplified EAST model containing concepts from the EAST-ADL structure, timing and requirements packages

superclass. Similarly, AllocationTarget is an abstract superclass which specifies to which elements an AllocatableElement can be allocated. HardwareComponentPrototype and HardwarePortConnector elements are defined as extensions of the AllocationTarget class. Within the proposed methodology, the concepts from the function, hardware and allocation models are used by the model transformations for the automatic generation of RCM models, as described later.

In this work, we show an instantiation of MoVES focusing on the verification of timing requirements. Therefore, let us see how the timing requirements and properties are modelled within EAST-ADL. The EAST-ADL timing package contains concepts for modelling the timing constraints stemming from the extra-functional requirements. Within EAST-ADL, a timing constraint is modelled by means of *TimingConstraint* elements, which are associated to DesignFunctionPrototype elements. The specification of the timing constraints is realised using two *Event* elements, which mark the scope of the related timing constraint and are contained within an *EventChain* element. The EventChain and Event elements are used by the proposed methodology for the specification of automatically generated timing constraints in the RCM models.

The requirement package offers means for describing the properties that the vehicular embedded system has to possess and their verification. To this end, the requirement package is further divided into two sub-packages which are *UseCases* and *VerificationValidation*. MoVES leverages specific concepts from the latter only. Within the VerificationValidation package, the *VVCase* elements represent concrete test activities which are associated to theDesignFucntionPrototype elements. A VVCase is modelled in terms of the *VVProcedure* and an *VVLog* elements which represent the adopted verification and validation technique and its description, respectively. A VVLog element is modelled in terms of a *VVActualOutcome* element, which specifies the actual output of the verification and validation activity. The proposed methodology uses the VVLog elements for back propagating the RCM models together with their timing verification results to the related EAST-ADL model.

Figure 1 shows a simplified EAST-ADL model represented as a block diagram using the EAST-ADL concepts discussed above. The model depicts a DesignFunctionPrototype called *Function*, which is allocated to a *HardwareComponentPrototype* called ECU. The Function and ECU elements are typed to the related DesignFunctionType and HardwareComponentType, respectively. Accordingly, ECU is an atomic node while Function is composed by two sub-functions called *SubFunction_1* and *SubFunction_2* connected via a FunctionConnector called *Connector*. Additionally, Function is associated with a VVCase called *Case* and a timing constraint called *Timing-*

Constraint.

B. RCM

RCM is a modelling language for the predictable development of resource-constrained embedded real-time systems developed by Arcticus Systems[4] in collaboration with Mälardalen University. With respect to the EAST-ADL structural abstraction levels, RCM acts at the implementation level and it is currently used by several OEM, Tier-1 and Tier-2 companies (e.g., Volvo Construction Equipment, BAE Systems Hägglunds, Hoerbiger and Knorr Bremse) in the vehicular domain. In its current definition, RCM provides support for the modelling of the software architecture, the execution platform, the allocation information and the timing properties of vehicular embedded systems[5] [12]. Within RCM, the embedded software architecture is modelled by means of the *Software Function* (SWC) elements and interactions among them. An SWC is the lowest-hierarchical element, which encapsulates basic software functions, and is defined by a *Behaviour* and an *Interface* elements. The Interface element is responsible for grouping the ports of a SWC. As RCM distinguishes between data and control flows among SWCs, an interface element contains two kinds of port: data and control. The interactions among SWCs are modelled by means of *Connector* elements. SWCs can be grouped in *Assembly* elements for constructing the software architecture in a hierarchical manner. SWCs and Assembly elements are contained in *Mode* elements, which are means for distinguishing different states or conditions in a system. For example, a system can execute the start-up mode when bootstrapping and afterwards shifts to the operational mode. Mode elements are contained within *Application* elements, which represent independent software functionalities of the system. Application elements provide means for isolating different software functionalities as well as for specifying the safety-criticality level in accordance to the ISO 26262 standard for the functional safety of road vehicles [13]. We refer to the RCM software models as the RCM models which contain the modelling elements for representing the software architecture, only. As the main goal of RCM is to provide support for the development of predictable vehicular embedded systems, timing properties and constraints are pivotal in the language, and they can be specified at different hierarchical levels of the software architecture (Application, Mode, Assembly). Timing constraints are modelled by *Timing Constraint* elements which are specified on the data ports of the related software element. Within RCM, the execution platform of the vehicular embedded system under development is modelled in terms of *Node*, *Core* and *Partition* elements. A Node element models the specific processor architecture and defines a unique run-time environment for the software architecture. A Node element contains one or more Core elements, which model the processing or computing unit of a Node element. Similarly, Core elements can contain one or more Partition elements, which represent the logical division of a Core elements into multiple computing resources. We refer to the RCM execution platform models as the RCM models which contain the modelling elements for representing the execution platform, only. Allocation information is modelled by means of the *isAllocated*

relation specified between any two *Allocatable* and *Allocator* elements. Allocatable is an abstract superclass which specifies the RCM software elements that can be allocated. Application, Mode, Assembly and SWC elements are defined as extensions of the Allocatable superclass. Similarly, Allocator is an abstract superclass which specifies to which execution platform element an Allocatable element can be allocated. Node, Core and Partition elements are defined as extensions of the AllocationTarget class.

Figure 2 shows a simplified RCM model represented as a block diagram using the RCM concepts described above. In particular, the model depicts a System called *System* which consists of a Node called *ECU* and an Application called *Application*. ECU is modelled as a single-core and single-partition node. Application is modelled by means of a Mode called *Operational* which contains an Assembly called *Function*. The internal architecture of the Assembly consists of two SWCs, called *SubFunction_1* and *SubFunction_2*, connected by two Connector elements, called *ConnectorData* and *ConnectorTrig*, for the data and the control flows, respectively. A TimingConstraint element is specified between the data output port of SubFunction_1 and the data input port of SubFunction_2.

C. Timing Analysis

Many vehicular embedded systems are constrained by stringent timing requirements that must be satisfied during their development. End-to-end timing analysis is a well-established technique to verify the timing requirements that are specified on these systems. Such an analysis must be integrated to the tool chain that is used for the model- and component-based development of these systems. In order to support the timing analysis an appropriate system view, called end-to-end timing model, should be extracted from the software architecture of the system under analysis. The end-to-end timing model consists of two models, namely timing model and linking model. The timing model includes the timing properties (e.g., priorities, periods, worst- and best-case execution times, offsets and jitter) and timing requirements (e.g., deadlines and delay constraints) regarding all tasks, messages and task chains in the distributed embedded system. On the other hand, the information about links, dependencies, control flows (activation information) and data flows (information regarding data exchanges) among tasks and messages in all task chains are captured in the linking model. For example, consider a task chain shown in Figure 3. The chain is distributed over three nodes that are connected by a network. The system timing model includes all the timing information (discussed above) in the three nodes and the network. Whereas, the linking model includes all the linking information (discussed above) in the chain that initiates at Task1 in the Sensor Node and terminates at Task4 in the Actuation Node. It should be noted that the end-to-end timing model discussed above is in line with the classical timing model for distributed embedded system [14], [15], [16]. The analysis engines use the end-to-end timing model to analyse the timing behavior of the system. In this paper we consider the end-to-end timing analysis given in [15], [17]. The analysis has been implemented in several industrial tools, e.g., [17]. The analysis results consist of response times of tasks and messages as well as system utilization. The analysis also calculates end-to-end response times and delays.

---

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

[5] Please note that the complete explanation of the RCM language is outside the scope of this work. The interested reader can refer to [12].
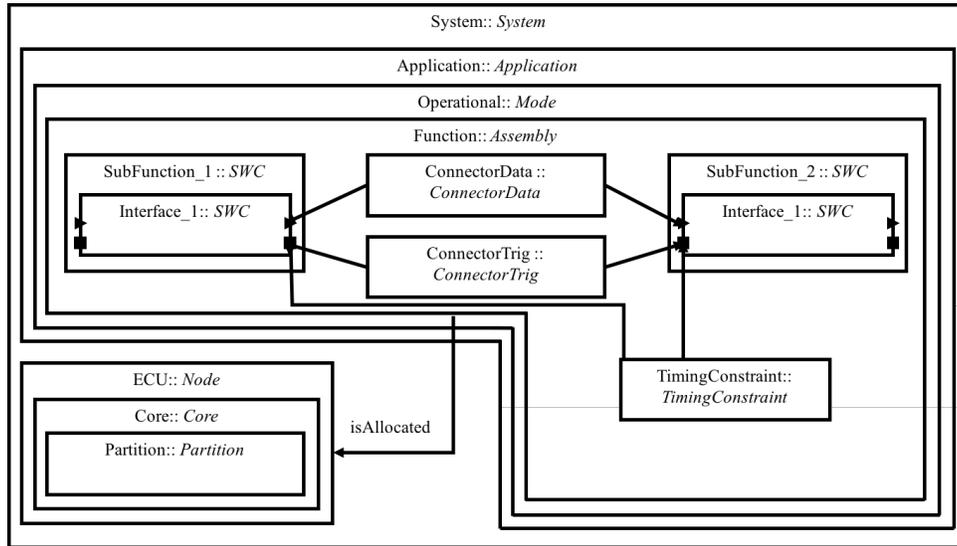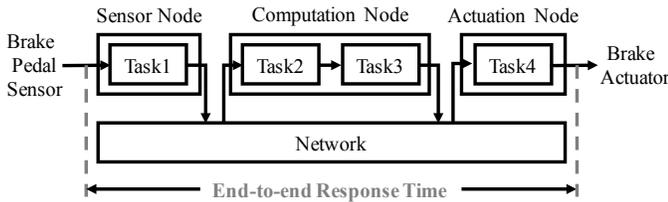
Fig. 2: Simplified RCM model

Fig. 3: Example showing end-to-end response time

Fig. 4: A task chain with independent activations of tasks

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 at the sensor node and the corresponding response, e.g., the brake actuation signal at the actuation node as shown in Figure 3. If the tasks in a chain are activated independently (e.g., by periodic clocks) then various types of end-to-end delays must also be computed to verify the timing behavior of the system. *Age* and *Reaction* are two such delays that are commonly found in vehicular embedded systems. The age delay in a task chain corresponds to the freshness of the data that is available at the output of the chain. This delay finds its importance in the control systems domain in vehicles. Whereas, the reaction delay in a task chain corresponds to the first reaction at the output of the chain for a given stimulus at the input of the chain. This delay finds its application in the body electronics domain in vehicles. In order to explain the age and reaction delays, consider a task chain in a single-node system as shown in Figure 4.

The chain consists of two tasks, namely τ_1 and τ_2. The tasks are activated by independent clocks of periods 25 milliseconds and 5 milliseconds respectively. Assume that the Worst-Case Execution Times (WCETs) of these tasks are 2 milliseconds and 1 millisecond respectively. Task τ_1 reads data from register Reg-1 and writes data to Reg-2. Similarly, task τ_2 reads data from Reg-2 and writes data to Reg-3. Since, the tasks are activated independently with different clocks, there can be multiple outputs (Reg-3) corresponding to any single
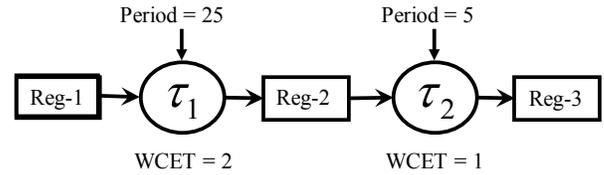
input (Reg-1) to the chain as shown by several uni-directional arrows in Figure 5.
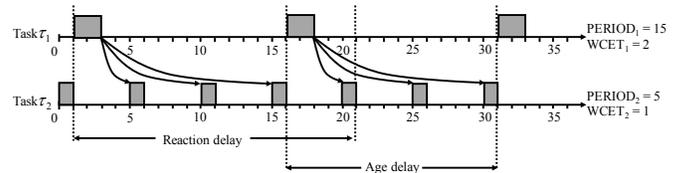

Fig. 5: Example showing end-to-end delays

The age and reaction delays are also identified in Figure 5. These delays are equally important in distributed embedded systems.

D. Paper Contributions in Relation with Authors' Previous Work

In this work, we present MoVES, a model-driven methodology for real-time distributed vehicular systems on single- and multi-core, supporting the development and architectural exploration of system designs with temporal awareness. The methodology leverages the interplay of EAST-ADL and RCM and consists of a fully automated mechanism defined in terms of a set of model transformations. RCM was originally thought for providing modelling purposes, but it did

(a) Development process without MoVES

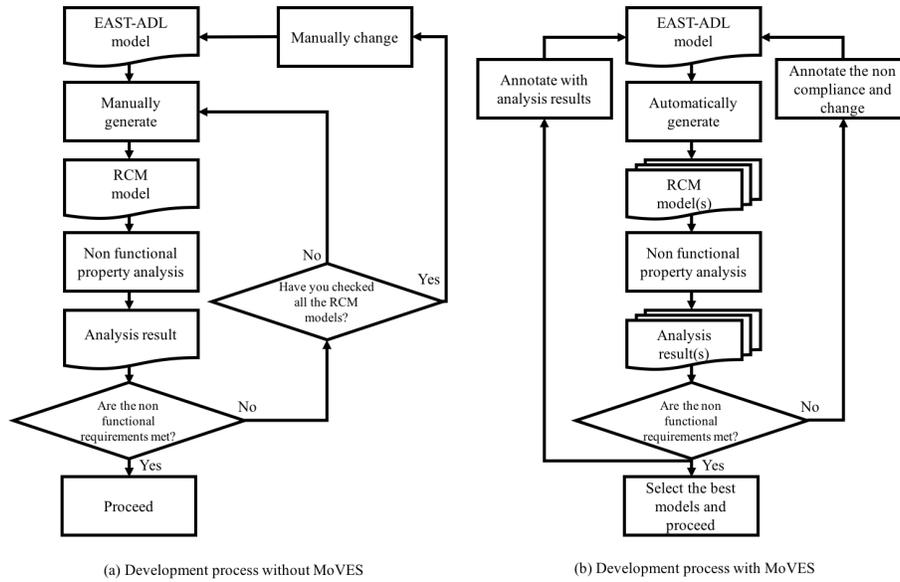(b) Development process with MoVES

Fig. 6: Comparison between a development process without (a) and with MoVES (b)

not feature a canonical definition of the language in terms of a metamodel. In [18], we reverse-engineered the RCM language and presented a preliminary metamodel definition of the RCM core elements. In [19], we provided a complete metamodel definition for RCM for modelling and timing analysis of vehicular embedded systems on single-core. In [12] we extended the RCM metamodel definition for the development of vehicular systems on multi-core. This extension introduced modelling elements for the representation of the execution platform and the allocation information. In this paper, we leverage an even more enhanced version of the RCM metamodel definition given in [12], which includes the concept of application. Early timing verification is paramount in the vehicular domain. To this end, in [20] we proposed a mechanism for the exploitation of RCM timing capabilities at EAST-ADL design level. Such a mechanism leveraged the RCM metamodel definition given in [19] and consisted of a single model transformation for the generation of the RCM software architecture and timing properties from an EAST-ADL model, only. Moreover, the mechanism in [19] can not be applied for the development of vehicular embedded systems on multi-core. In this paper, by exploiting the new modelling capabilities of RCM, we introduce a methodology for the development of real-time distributed vehicular embedded systems on single- and multi-core, which guides the engineer to viable solutions with respect to timing requirements. To this end, the presented methodology leverages a refined version of the model transformation described in [20] and introduces i) four new model transformations for the automatic translation of the EAST-ADL FunctionalDesignArchitecture, Hardware-DesignArchitecture and Allocation to RCM models and ii) a new model transformation that captures the timing analysis results and the corresponding RCM models and propagates them to the design level. Finally, we discuss the applicability of the proposed methodology by leveraging an industrial running example.

III. THE MOVES METHODOLOGY: WHY?

Among other factors, early verification of non functional requirements can positively affect the cost-efficiency of the software development for vehicular real-time embedded systems. Currently, early verification of non functional requirements is hard to achieve due to the lack of automation supporting models integration and analysis. For instance, let us consider a typical development process as described by the flowchart in Figure 6 (a).

As meaningful non functional analysis (such as timing) must be run on implementation models, the engineer is required to create a RCM model manually. The non functional analysis of interest is run on the manually created RCM model and the result is verified against the given set of non functional requirements. If the specified requirements are not met, the engineer is required to iterate the process and modify or create a new RCM model until a compliant one is found. Since the process of creating and verifying implementation models is expensive, it is not leveraged early in the development process for having quick and early feedback on the design level models. To boost early verification, in this paper we propose MoVES, a novel model-driven methodology for the development of vehicular real-time embedded systems supporting early verification of non functional properties.

Let us consider a development process equipped with the proposed methodology as described by the flowchart in Figure 6 (b). In this setting, all meaningful RCM models are automatically generated from the design models and analysed by means of model transformations. Given a set of non functional requirements, model transformations are responsible for the selection and back propagation of the best RCM model (or set of models), too. Besides relieving the engineer from the manual and iterative definition of a RCM model, the proposed methodology enables early verification at design level. Moreover, while several iterations may be needed in the manual process to reach a RCM model that fulfils non functional

requirements, MoVES is able to generate all meaningful RCM models and identify the best one(s) (as shown in Section VI) automatically in one single iteration.

IV. MoVES for Timing

Timing requirements are crucial for our domain of interest, vehicular real-time embedded systems, and that timing-related issues are typical problems arising very late in the development. For this reasons, in this work we discuss an instantiation of MoVES for supporting the development and architectural exploration of system-designs with temporal awareness.

MoVES leverages the interplay of EAST-ADL and RCM and provides automation for their integration by means of six model transformations. Figure 7 gives a graphical representation of the methodology and its composing tasks.

The first step of the methodology is the automatic generation of the RCM models representing the software architecture and its timing properties at the implementation level. Such a generation process is characterised by a one-to-many mapping meaning that multiple RCM software models can be a valid translation of an EAST-ADL FucntionalDesignArchitecture, where each of the RCM model would entail different timing and control flow information. Within MoVES, this generation is entrusted to the FDA2RCM model transformation. In particular, starting from an EAST-ADL FucntionalDesignArchitecture complemented with EAST-ADL timing requirements, FDA2RCM generates, in a single execution, the set of the corresponding RCM software models equipped with RCM timing constraints as opposite to a manual generation considering only one specific solution. The second step of the methodology is the automatic generation of the RCM model representing the execution platform at the implementation level. This step is performed by the HDA2RCM model transformation which starts from an EAST-ADL HardwareDesignArchitecture and generates a corresponding RCM execution platform model. As RCM models the execution platform in a more detailed way than EAST-ADL (RCM employs the concepts of core and partition), the engineer can manually refine the generated RCM execution platform models. Detailed execution platform models are pivotal for the specification of the allocation information which, in turn, affects timing analysis. The third step of the methodology merges the RCM software and execution platform models into complete RCM models, where the allocation information can be translated from the EAST-ADL Allocation. To this end, the MERGE model transformation is responsible for merging each generated RCM software model with the generated RCM execution platform model. The result is a set of complete RCM models. The fourth step of the methodology is the generation of the allocation information on the complete RCM models and it is entrusted to the A2RCM model transformation. In particular, A2RCM is responsible for the translation of the allocation information from an EAST-ADL Allocation model to the RCM complete models generated as a result of the MERGE transformation. Since RCM leverages a more fine-grained allocation mechanism than EAST-ADL, the methodology, by means of the ALLOCATION model transformation, is able to generate additional RCM allocation configurations that can not be derived from EAST-ADL. At this point, the RCM models contain all the information needed for the model-based timing analysis. Once the timing analysis

is run[6], the analysis results are produced and collected. The last step of the methodology is the back-propagation of the analysis results at the design level for enabling timing-aware design decisions and it is performed by the BP transformation. To this end, BP enriches the initial EAST-ADL model with the analysis results and the related RCM models such that the engineer can take timing-aware design decision on the EAST-ADL models without creating or nor editing RCM models.

In the following sections, we present a detailed discussion of each of the above mentioned model transformations. The complete implementation of the proposed methodology is available at http://www.mrtc.mdh.se/MoVES/.

A. FDA2RCM

FDA2RCM is a one-to-many model-to-model transformation between EAST-ADL and RCM, which is realised in the Eclipse Modeling Framework (EMF)[7] using the Janus Transformation Language (JTL) [9]. JTL is a constraint-based bidirectional model transformation language specifically tailored to support non-determinism by generating all the possible target models in a single execution. Its implementation relies on the Answer Set Programming (ASP) [21], which is a type of declarative programming able to address hard (primarily NP-hard) search problems and based on the model (answer set) semantics of logic programming. The ASP solver is responsible to find and generate, in a single execution, all the possible target models that are consistent with the transformation rules following a deductive process. JTL adopts a QVTr-like syntax and allows a declarative specification of relationships between MOF models. It supports object pattern matching and automatically creates traces information to record what occurred during a transformation execution.

The FDA2RCM transformation is the starting point of the methodology and provides for the translation of the software architecture of the vehicular embedded system under development and its timing properties. An initial version of the FDA2RCM transformation is given in [20]. The proposed version i) extends the one in [20] with new rules for the translation of the EAST-ADL FunctionalDesignArchitecture elements into the RCM software elements composing the current RCM structural hierarchy (e.g., System, Application, etc.) and ii) replaces the logic constraints in the pre- and post-conditions of the transformation rules in favour of more compact and understandable transformation rules. In a nutshell, the FDA2RCM transformation is responsible for translating the elements of the EAST-ADL FunctionalDesignArchitecture to RCM software elements. In particular, it maps the EAST-ADL DesignFunctionPrototype, FunctionPort, FunctionConnector, AgeConstraint and ReactionConstraint elements to the RCM Assembly, SWC, Port, ConnectorData, DataAge and DataReaction elements, respectively. Additionally, the FDA2RCM transformation provides for the automatic generation of the RCM ConnectorTrig, Clock and Sink elements representing detailed control flow and timing information. As detailed control flow and timing information is not described in the EAST-ADL FunctionalDesignArchitecture, a single source

---

[6]The proposed methodology leverages model-based timing analysis. However, the analysis itself is not part of the contributions of this work. The interested reader can refer to [17] for further details.
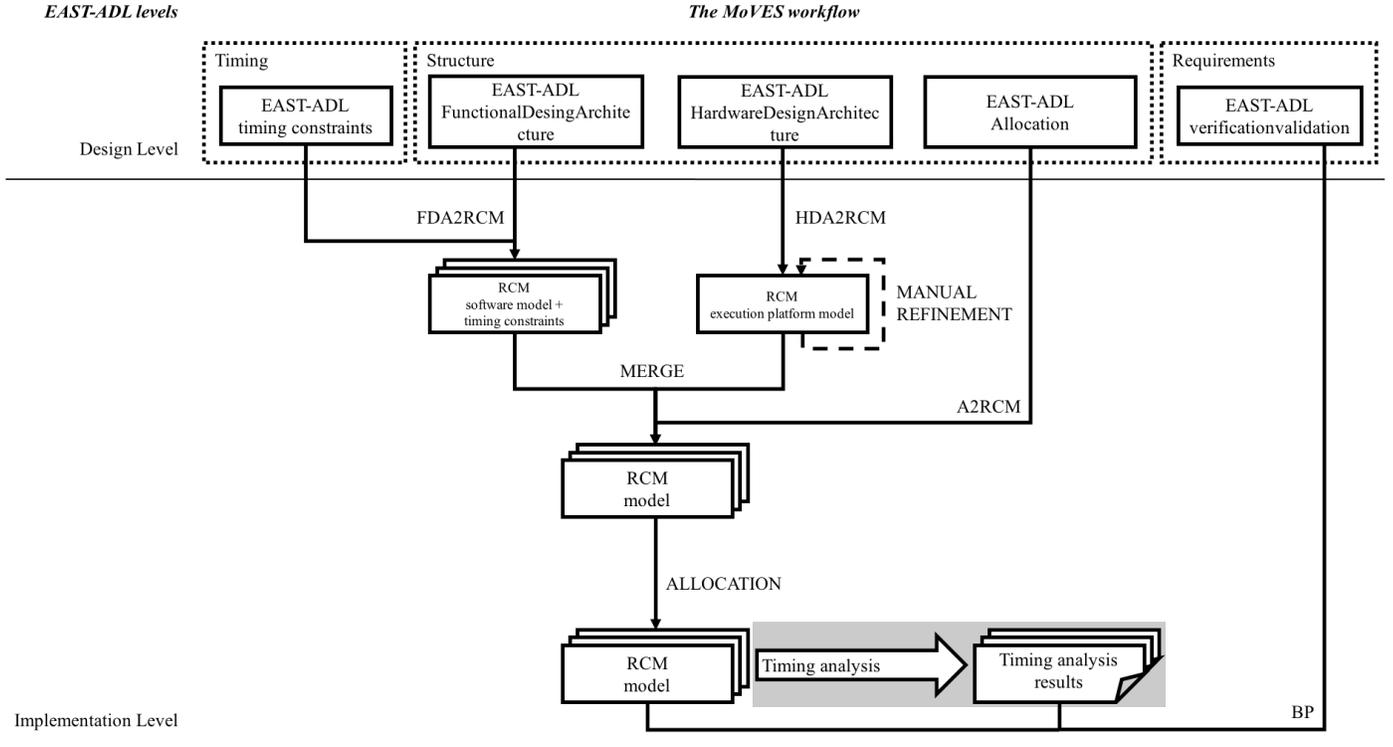
[7]https://eclipse.org/modeling/emf/

Fig. 7: The MoVES and its composing tasks

EAST-ADL FunctionalDesignArchitecture can not be univocally translated in a single RCM model. For instance, the DFP2C and the DFP2CCS rules in Listing 1 define a non deterministic portion of the FDA2RCM transformation where a DesignFunctionPrototype element can be translated either to a SWC element or to a SWC element equipped with a Clock and a Sink element.

```
1 transformation FDA2RCM(dl:designLevel, rcm:RubusMM) {
2  relation DFP2C {
3   name, id:String;
4   checkonly domain dl ps:designLevel::
        DesignFunctionPrototype {
5    name = name,
6    id = id,
7    type = t:designLevel::DesignFunctionType {
8     isElementary = true
9    }
10    };
11    enforce domain rcm at:RubusMM::Assembly {
12     circuit = c : RubusMM::Circuit {
13      name = name,
14      id = id
15     }
16    };
17    where{...}
18  }
19  relation DFP2CCS {
20   name, id:String;
21   checkonly domain dl ps:designLevel::
        DesignFunctionPrototype {
22    ...
23    };
24    enforce domain rcm at:RubusMM::Assembly {
25     circuit = c:RubusMM::Circuit {
26      name = name,
27      id = id,
28      interface = i:RubusMM::Interface {...}
29     },
30    clock = clk:RubusMM::Clock {...},
31    sink = snk:RubusMM::Sink {...},
```

```
32    connectorTrig = con1:RubusMM::ConnectorTrig {...},
33    connectorTrig = con2:RubusMM::ConnectorTrig {...}
34   };
35    where{...}
36  }
37 }
```

Listing 1: Fragment of the FDA2RCM model transformation in JTL.

In this context, the JTL engine is able to generate, in a single execution, all the RCM software models entailing different and unique configurations of, e.g., SWC, Clock and Sink elements as opposite to a manual translation considering only a specific model. It is important to notice that, logic constraints can be applied for narrowing the space of the generated models. For instance, the execution of the FDA2RCM transformation to the source EAST-ADL model depicted in Figure 8 (a)[8] could be narrowed by means of logic constraints which could guide the generation of RCM models to those entailing valid configurations of SWC, Clock and Sink elements, only.

Accordingly, only the two RCM models in Figure 8 (b) and Figure 8 (c) would be generated. In the former, the SWC Actuator is activated from the SWC Sensor through the Connector Connector_Trig, while in the latter it is activated by the independent Clock Clock_Actuator.

B. HDA2RCM

HDA2RCM is a model-to-model transformation between EAST-ADL and RCM, which is realised by means of JTL.

---

[8]For better understandability, we represent EAST-ADL and RCM models by means of a simplified graphical concrete syntax.

(a) Source EAST-ADL model

(b) Target RCM model

(c) Target RCM model

Fig. 8: Example of simplified source and target models for the FDA2RCM transformation
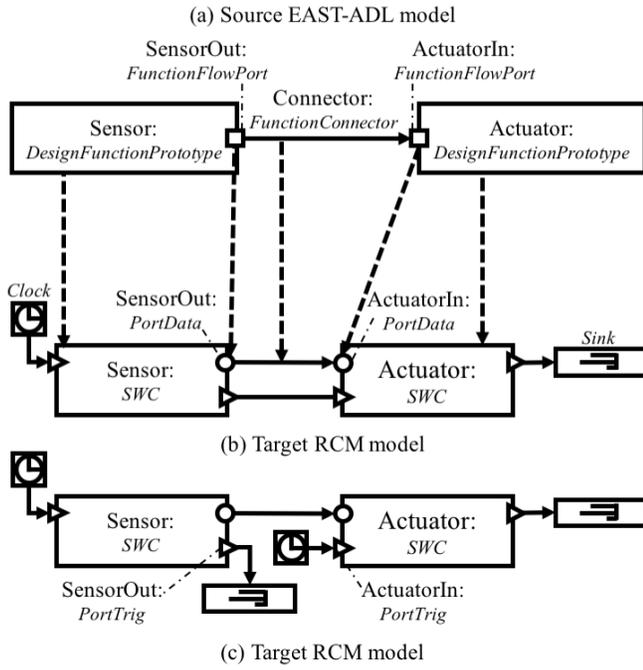

(a) Source EAST-ADL model

(b) Target RCM model

Fig. 9: Example of simplified source and target models for the HDA2RCM transformation

Together with the FDA2RCM transformation, HDA2RCM is the staring point of the methodology and provides automation means for the translation of the execution platform of the vehicular embedded system under development. In fact, execution platform models are pivotal for the specification of the allocation information which, in turns, affects timing analysis. Trivially, the over utilisation of a processor or a core can lead to timing deadline misses.

```
1  transformation HDA2RCM(dl:designLevel, rcm:RubusMM) {
2   relation AtomicHardwareComponentPrototype2Node {
3    id, name: String;
4
5    checkonly domain dl hardwareComponentPrototype :
           designLevel::HardwareComponentPrototype {
6     id = id,
7     name = name,
8     type = hardwareComponentType : designLevel::Node {...}
9    };
10
11   enforce domain rcm system : RubusMM::System {
12    connectorNetwork = connectorNetwork : RubusMM::
           ConnectorNetwork {},
13    node = node : RubusMM::Node {
14     id = id,
15     name = name
16     core = core : RubusMM::Core {
17      ...
18      partition = partition : RubusMM::Partition {...}
19     }
20    };
21    where {...}
22   }
23 }
```

Listing 2: Fragment of the HDA2RCM model transformation in JTL

The HDA2RCM transformation is responsible for translating the EAST-ADL HardwareComponentPrototype elements to RCM execution platform elements. In particular, it maps the EAST-ADL Node, HardwarePortConnector and Hardware-
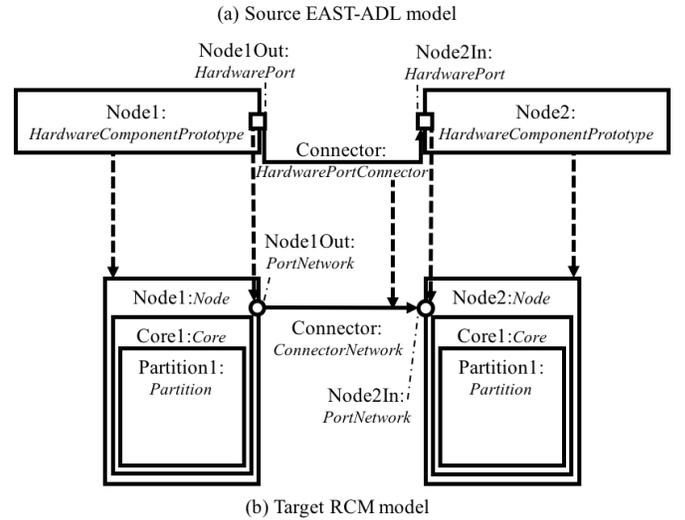
Port elements to the RCM Node, ConnectorNetwork and Port elements, respectively. As discussed in Section II, RCM models the execution platform with a structural hierarchy of elements consisting of Node, Core and Partition. However, EAST-ADL provides modelling element for the representation of Node elements, only. Therefore, in order to generate valid RCM models, the HDA2RCM transformation automatically generates, for each RCM Node element, a Core and a Partition element, too. Please note that, the engineer can still manually refine the generated RCM execution platform model, if needed. Listing 2 depicts an extract of the HDA2RCM transformation consisting of the transformation rule responsible for the generation of RCM Node, Core and Partition elements. Figure 9 depicts an example of an execution of the HDA2RCM model transformation. In particular, the EAST-ADL model depicted in Figure 9 (a) and consisting of two connected Node elements, Node1 and Node2, is translated into the RCM model depicted in Figure 9 (b) consisting of two connected Node elements, Node1 and Node2, containing a Core and a Partition element each.

C. MERGE and A2RCM

MERGE and A2RCM are two model-to-model transformations realised within EMF using the QVT Operational (QVT-O) language [22]. Query/View/Transformation (QVT) is a standard set of model transformation languages defined by the Object Management Group and it is composed by three model transformation languages, which are QVT-O, QVT Relations and QVT Core. QVT-O is an imperative language especially designed for writing unidirectional model transformations when declarative model transformations are hard to specify due to the absence of direct correspondence between elements of the source and target models. Thereby, a QVT-O model transformation explicitly specifies the steps to execute in order to generate a target model starting form a source one.

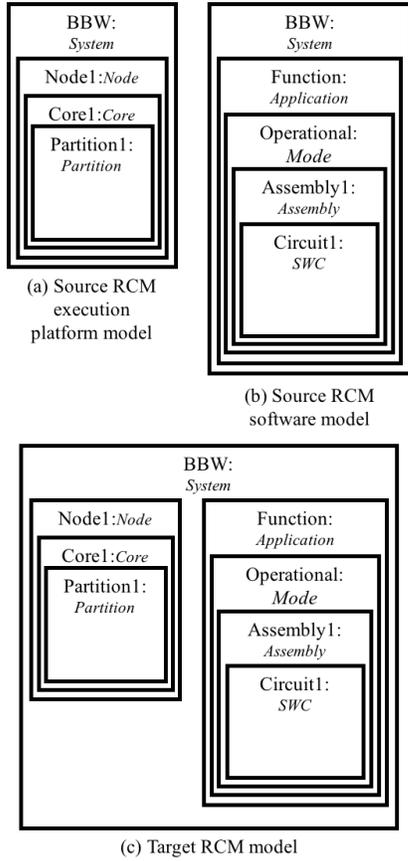Once the FDA2RCM and the HDA2RCM transformations

Fig. 10: Example of simplified source and target models for the MERGE transformation

Fig. 11: Example of a simplified source and target models for the A2C transformation

are run, a set of RCM software models and one RCM execution platform model are produced. In this context, the MERGE transformation is responsible for merging a RCM software model to the RCM execution platform model with the purpose of creating a complete RCM model where the allocation information can be translated and refined. In a nutshell, the MERGE transformation performs a weaving of the RCM models, where the modelling elements of the RCM execution platform model are linked to the System element of the RCM software model through its Node reference. Let us consider the RCM execution platform and software models depicted in Figure 10 (a) and Figure 10 (b), respectively. The former consists of a System element called BBW containing a Node element called Node1, which contains a Core element Core1. Eventually, Core1 contains a Partition element called Partition1. The latter consists of a System element called BBW, which in turns contains an Application element called Function. Function contains a Mode element called Operational, which contains an Assembly element called Assembly1. Assembly1 contains a SWC called Circuit1. Accordingly, the application of the MERGE transformation would produce the RCM model depicted in Figure 10 (c) where the RCM execution platform elements in Figure 10 (a) are integrated into the RCM software model in Figure 10 (b) by means of the Node reference of the System BBW element. A2RCM is an in-place transformation [23] which follows the MERGE transformation and it is responsible for translating the allocation information from the EAST-ADL
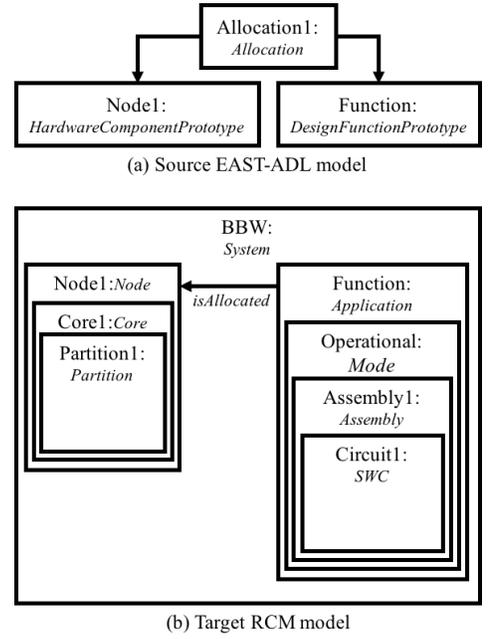
Allocation to each of the RCM models generated from the MERGE transformation. Allocation information is crucial for the model-based timing analysis as, e.g., the over utilisation of a node or a core element can result in violating the timing requirements. As discussed in Section II, in EAST-ADL the allocation information is modelled by means of a set of Function Allocation elements which links the DesignFunctionPrototype element to the HardwareComponentPrototype element. For instance, the Function Allocation Allocation1 in Figure 11 (a) allocates the DesignFunctionPrototype Function to the HardwareComponentPrototype Node1. In RCM, the allocation information is specified by means of the isAllocated reference of the Allocatable elements. Therefore, the A2RCM transformation is responsible for setting the RCM isAllocated references according to the EAST-ADL Function Allocation elements. For instance, if we apply the A2RCM model transformation to the EAST-ADL Allocation depicted in Figure 11 (a), we obtain the RCM model in Figure 11 (b). Accordingly, the isAllocated reference of the Application element called Function is set to the Node element called Node1.

D. ALLOCATION

ALLOCATION is a one-to-many, in-place model transformation on the RCM language realised by means of JTL. As discussed in Section II, compared to EAST-ADL, RCM leverages a more fine-grained structural hierarchy for the modelling of the execution platform and the allocation. In particular, within RCM, any Allocable element (Function, Mode, Assembly and SWC) can be allocated to any of the Allocator element (Node, Core and Partition). Due to the different granularity between RCM and EAST-ADL, complete allocation information can not be directly translated from an EAST-ADL Allocation. In this context, the ALLOCATION transformation provides automation means for the generation

of the allocation information in the RCM models when a direct translation from EAST-ADL is not possible. In other words, the ALLOCATION transformation automatically generates the isAllocated reference of the RCM Allocatable elements and sets it to any of the RCM Allocator elements. As there can be several allocation strategies, the engineer is required to express a choice over the preferred one. This can be done by toggling the comments on the transformation rules which realise the desired allocation strategy (e.g., Assembly to Core, Assembly to Partition, SWC to Core, etc.). Based on the user choice, the ALLOCATION transformation is able to generate, in a single execution, all the RCM models which entail different and unique allocation configurations. Similarly to the FDA2RCM, logic constraints can be applied for narrowing the number of the generated RCM models. This is particularly useful when partial allocation information is already available (as in case of, e.g., legacy vehicular systems) or for discarding specific allocation configurations.

```
1 transformation ALLOCATION(source:RubusMM, target:RubusMM) {
2
3  relation Assembly2AllocatedAssemblyNode{
4   name, id: String;
5
6   checkonly domain source as:RubusMM::Assembly {
7       name = name,
8       id = id
9       };
10
11      enforce domain target at:RubusMM::Assembly {
12       name = name,
13       id = id,
14       isAllocated = n:RubusMM::Node {...}
15       };
16 }
17 }
```

Listing 3: Fragment of the ALLOCATION model transformation in JTL.

Listing 3 shows a fragment of the ALLOCATION transformation which depicts the transformation rule responsible for allocating the Assembly element to Node element. Accordingly, the JTL engine generates, in a single execution, all the RCM models entailing different combinations of Assembly to Node elements. That is, the application of the ALLOCATION transformation to the RCM model depicted in Figure 12 (a) generates the two RCM models depicted in Figure 12 (b) and Figure 12 (c). In the former, Assembly1 is allocated to Node1, while in the latter Assembly1 is allocated to Node2.

E. BP

BP is an in-place, text-to-model transformation on the EAST-ADL metamodel realised by means of QVT-O. Within MoVES, BP is the last step for unveiling the RCM models and their related analysis results at the design level thus for enabling timing-aware design decisions. In fact, once the RCM model and the analysis results are unveiled at design level, the engineer can easily grasp the compliance of the starting EAST-ADL models to the specified timing requirements. Even further, she can select the most appropriate RCM model, among the compliant ones, for proceeding with the development process. The BP transformation uses the EAST-ADL VVCase modelling element from the Requirement package. In particular, it automatically creates the VVLog and VVActualOutcome modelling elements for the given VVCase element and sets their attribute Date, ID and Name. Moreover, the BP
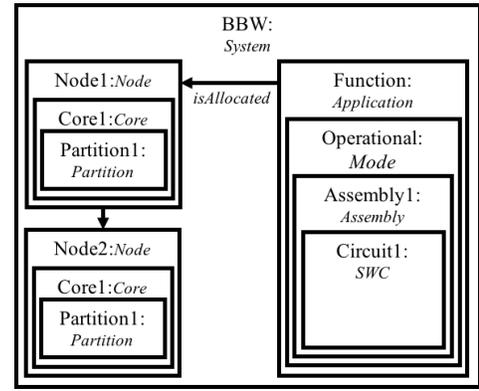
(a) Source RCM model
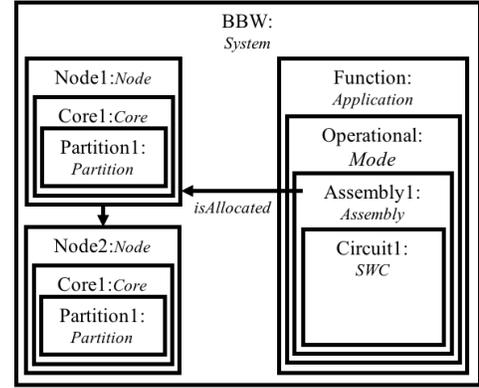
(b) Target RCM model
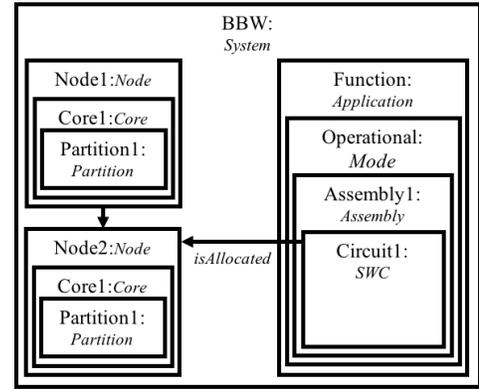
(b) Target RCM model

Fig. 12: Example of simplified source and target models for the ALLOCATION transformation

transformation is responsible for setting the Text attribute of the EAST-ADL VVACtualOutcome element with the URLs of the folders containing the timing analysis results and the related RCM models.

V. CASE STUDY

In this section, we demonstrate the usability of MoVES by developing the adaptive cruise control system as an extension of the cruise control system. The cruise control system is a vehicular feature which allows the vehicle to keep a steady speed to the value provided by the driver. To this end, it employs (at least) 4 modules: one for communicating/controlling

the engine, one for communicating/controlling the brakes, one for communicating with the driver's instrument cluster and one for the computation. However, the traditional cruise control system does not take into account traffic information such as presence of other vehicles or obstacles. The Adaptive Cruise Control (ACC) system is a vehicular feature which allows a vehicle's cruise control to adapt the vehicle's speed to the surrounding environment. More precisely, once the user sets a target speed and a time gap for the vehicle, a radar detects slow-moving vehicles or other obstacles that are in the path of the vehicle. In case an obstacle or a slower vehicle is detected, the ACC system slows down the vehicle or brakes to keep the desired distance between the ACC vehicle and the obstacle or the forward vehicle, where the distance is calculated as a function of the specified time gap and the speed of the vehicle. When the ACC system detects that the forward vehicle or the obstacle is no longer in the vehicle's path, it speeds up the vehicle to maintain the cruise speed set by the driver. With respect to the cruise control functionality, the ACC enhances the computation module with functionalities which provide the adaptive features. Figure 13 shows a block diagram of the ACC system[9] that is adapted from [24].
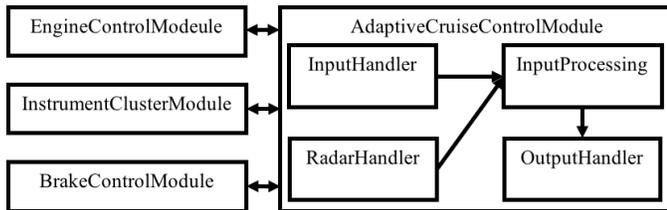

Fig. 13: Block diagram of the ACC system.

The InstrumentClusterModule is responsible for collecting the user's inputs, such as speed and time gap, and for sending them to the AdaptiveCruiseControlModule. The EngineControlModule and the BrakeControlModule are responsible for sending the information regarding speed and braking of the vehicle to the AdaptiveCruiseControlModule, respectively. AdaptiveCruiseControlModule is the core of the ACC feature and it is responsible for calculating the acceleration and the braking of the vehicle in the presence of forward vehicles or obstacles. In particular, the InputHandler software component is responsible for acquiring and processing the inputs coming from the other modules and to forward them to the InputProcessing software component. Similarly, the RadarInput software component is responsible for acquiring and processing the information coming from the radar and forwarding it to the InputProcessing software component. Based on the user inputs on the cruise mode and considering the presence of a forward obstacle, the InputProcessing software component is responsible for computing the decision whether the vehicle has to slow down, speed up or keep a steady speed. It communicates this information to the OutputHandler software component which, in turn, is responsible for sending the related brake torque and throttle signal to the BrakeControlModule and EngineControlModule, respectively.

According to MoVES, the development starts from an

[9]The interested reader can access the full ACC case study implementation at http://www.mrtc.mdh.se/MoVES/.

EAST-ADL FucntionalDesignArchitecture, HarwareDesignArchitecture and Allocation models. Figure 14 depicts an extract of the EAST-ADL FucntionalDesignArchitecture for the ACC. In the remainder of this section, for the sake of verbosity, we adopt a simplified concrete syntax both for the EAST-ADL and RCM models and omit some modelling elements if of no interest for the discussion.

The modules and their inner architecture are represented by means of the DesignFunctionPrototype and FunctionFlowPort elements. For instance, the RadarInput software component is represented by means of the RadarInput DesignFunctionPrototype element and the RadarIn and RadarOut FunctionFlowPort elements. The connections among software components and modules are represented by means of FunctionConnectorelements connecting the FunctionFlowPort elements. Fo instance, the RadarInput and InputProcessing DesignFunctionPrototype elements are connected by means of the RadarOut2RadarIn FunctionConnector connecting the RadarOut and RadarSignal FunctionFlowPorts. In addition to the architectural elements, two timing constraints, denoted by Reaction Constraint T1 and Age Constraint T2, and VVCase element, denoted by TimingANalysis, are specified. According to these constraints:

The reaction and age delays between the arrival of the radar signal at the input of the InputProcessing software component and its delivery to the BrakeControlModule shall not exceed 25 and 15 milliseconds, respectively. Starting from the EAST-ADL FucntionalDesignArchitecture in Figure 14, MoVES automatically generates RCM software models. In particular, RCM System, Application and Mode elements, namely AdaptiveCruiseControl, AdaptiveCruiseControl and AdaptiveCruiseControl_Operational are generated. The AdaptiveCruiseControlModule, EngineControlModule, BrakeControlModule and InstrumentClusterModule DesignFucntionPrototype elements are translated into RCM Assembly elements while the InputHandler, RadarInput, InputModeControl, Processing and Outputhandler DesignFunctionPrototype elements are translated into RCM SWCs. PortData, ConnectorData and TimingConstraint elements are generated from the EAST-ADL FunctionFlowPort, FunctionConnector and TimingConstraint elements, respectively. Due to the lack of control flow information in EAST-ADL, RCM PortTrig, ConnectorTrig, Clock and Sink elements are automatically generated by the FDA2RCM.

However, as there might be multiple ways of specifying these elements, this generation produces four RCM software models each of which entails different and unique combinations of RCM PortTrig, ConnectorTrig, Clock and Sink elements. For instance, let us consider the activation of the OutputHandler SWC in the two RCM models in Figure 15. In the RCM model in Figure 15 (a), the OutputHandler SWC is triggered by an independent clock whether in the RCM model in Figure 15 (b) it is triggered by its predecessor, InputProcessing. Within MoVES, the second step is the translation of the EAST-ADL HardwareDesignArchitecture model representing the execution platform architecture. Figure 16 depicts the EAST-ADL HardwareDesignArchitecture for the ACC feature.

It is realised by means of two Node elements, Node_1 and Node_2, each of which represents a MPC560XP microcontroller that is a single-core microcontroller for vehicular
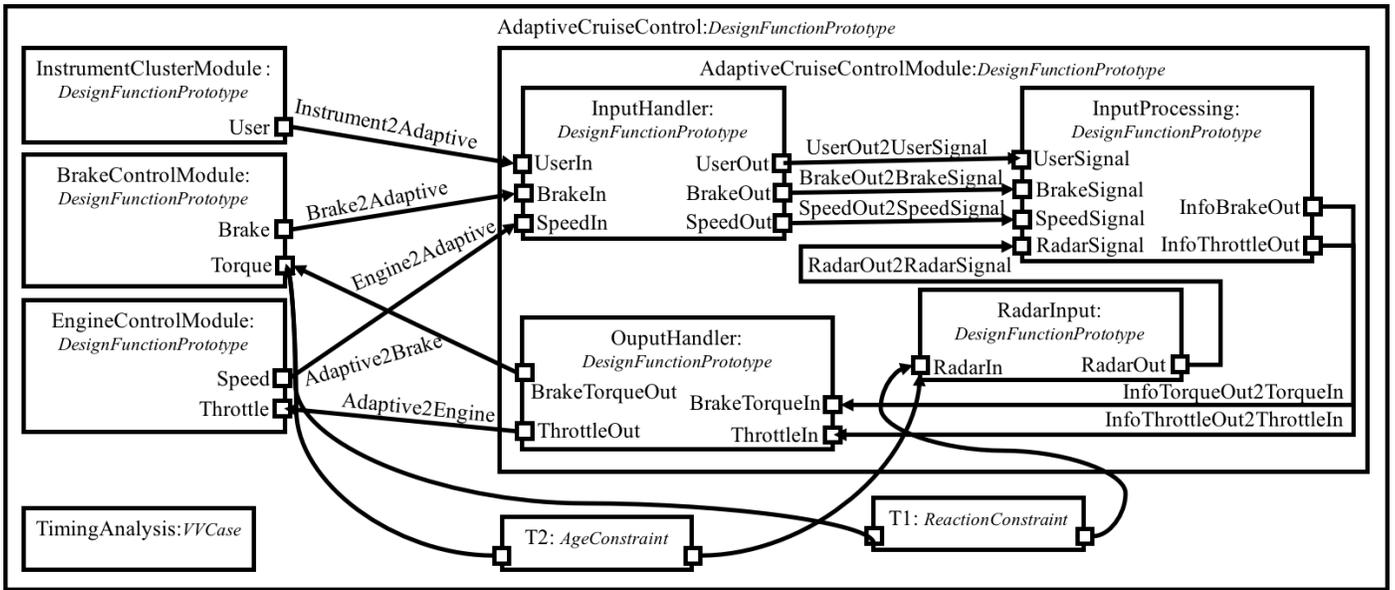
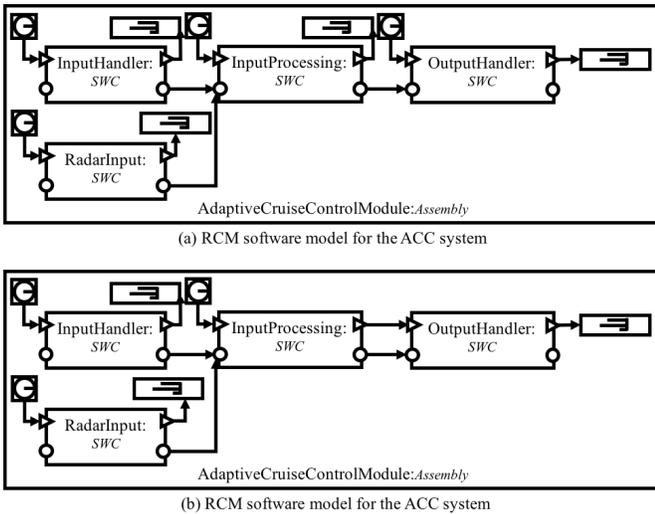Fig. 14: EAST-ADL FunctionalDesignArchitecture of the ACC system
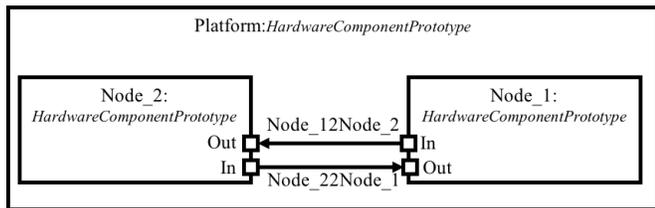

(a) RCM software model for the ACC system

(b) RCM software model for the ACC system

Fig. 15: Two of the four RCM software models of the ACC system

Fig. 16: EAST-ADL HardwareDesignArchitecture of the ACC system

and industrial safety applications[10]. Two HardwarePortCon-

nector elements, Node_12Node_2 and Node_22Node_1, re-alise the communication between the two nodes. Starting from the EAST-ADL HardwareDesignArchitecture in Figure 16, MoVES generates the RCM model depicted in Figure 17.
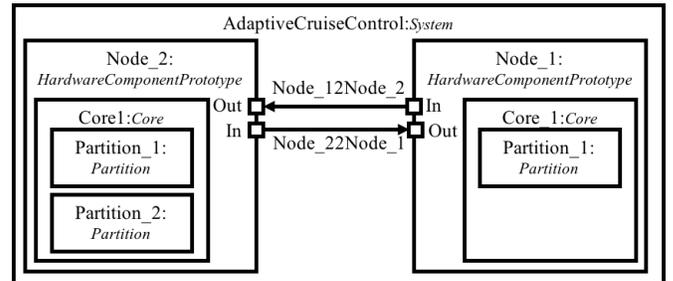

Fig. 17: RCM execution platform model of the ACC system

In particular, the EAST-ADL Node and HardwarePortConnector elements are translated into RCM Node and NetworkConnector elements. According to the HDA2RCM transformation, the inner architecture of each generated RCM model is enriched with a Core and a Partition element. That is, Core_1 and Partition_1 elements are generated for the RCM Node_1 and Node_2 elements. Eventually, by means of manual refinements, an additional Partition element, Partition_2 is added to the RCM Node_2. At this point, MoVES merges the RCM software and execution platform models and the result is a set of four complete RCM models where the allocation information from the EAST-ADL Allocation can be translated. To this end, Figure 18 depicts the EAST-ADL Allocation model for the ACC feature.

The EAST-ADL Allocation model consists of four FunctionAllocation elements mapping AdaptiveCruiseControlModule to Node_1 and InstrumentClusterModule, EngineControlModule and BrakeControlModule to Node_2. Accordingly, MoVES translates the allocation information on the RCM
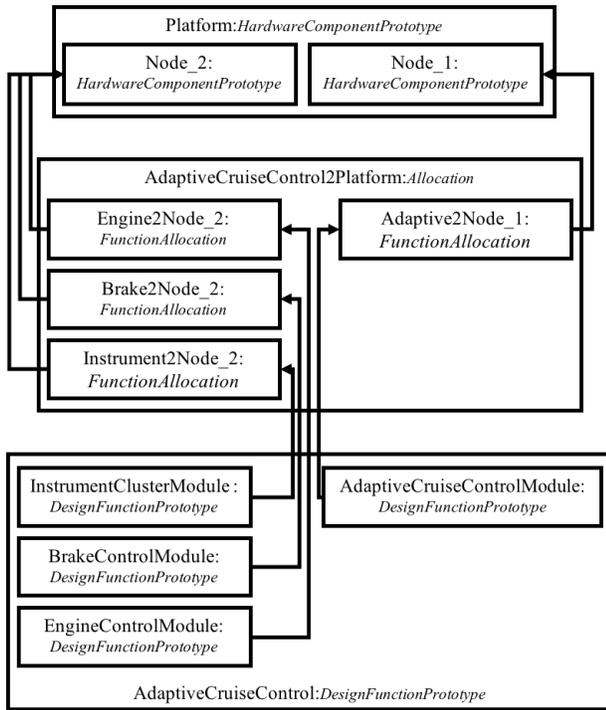
Fig. 18: EAST-ADL Allocation of the ACC system

models resulting from the MERGE transformation. Figure 19 depicts one example of RCM model along with the translated allocation information. Consequently, the isAllocated refer-
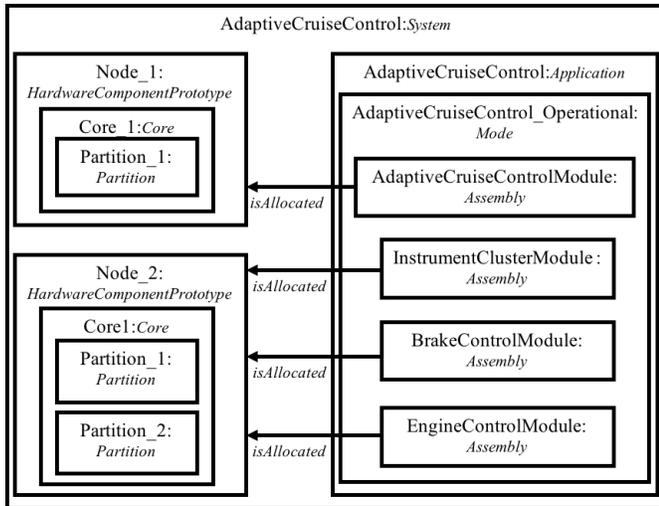

Fig. 19: RCM model for the ACC system with the allocation information

ences of the InstrumentClusterModule, BrakeControlModule and EngineControlModule assemblies are set to the Node_2 element while the isAllocated reference of the Adaptive-CruiseControlModule Assembly elements is set to the Node_1 element. However, as EAST-ADL does not leverage the concepts of cores and partitions for the modelling of the execution platform, the allocation of the InstrumentClusterModule, EngineControlModule and BrakeControlModule elements can

not be refined with respect to core and partition elements of Node_2. Nevertheless, such an information can be automatically generated from the ALLOCATION transformation as described in Section III. In particular, for the ACC system, we decided to choose an allocation strategy which allocates Assembly to Partition elements. Consequently, as there are 8 different ways to allocates the three Assembly elements to the two Partition elements, MoVES generates a final set of 32 RCM models (8 refined RCM models for each of the 4 RCM models resulting from the A2C). Eight of the 32 final RCM models are depicted in Figure 20.
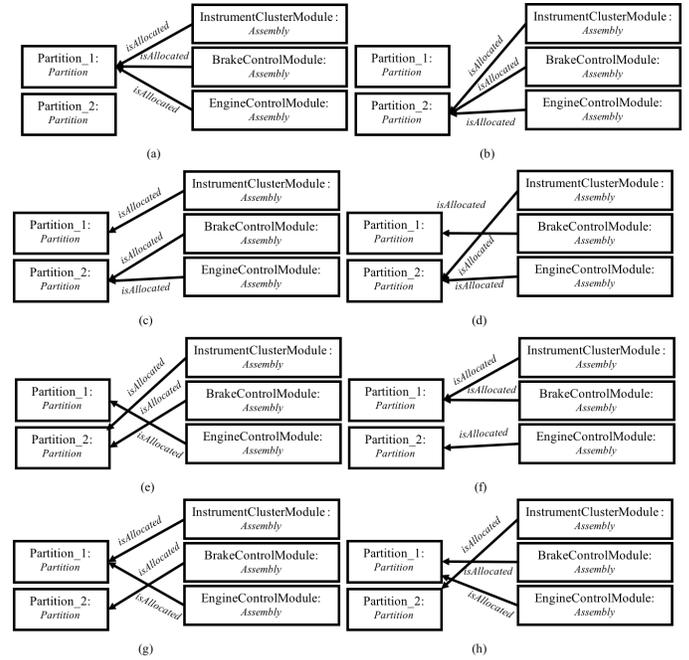

Fig. 20: RCM models for the ACC system with the refined allocation information

At this point, model-based timing analysis is run on each of the generated 32 RCM models with the aim of verifying whether the specified reaction and age constraints are met. Table I summarises the results of the timing analysis. Table I summaries the analysis results for the 32 generated RCM models. It can be seen that, out of the 32 RCM models of the ACC system, only 16 RCM models satisfy the specified age and reaction constraints. In particular, MoVES is able to automatically identify the 5 RCM models with the best timing performances. In this respect, it is important to note that MoVES can be easily constrained for notifying the engineer only with the best RCM models rather than with all the compliant ones. Eventually, the VVLog and VVActualOutcome elements are created for the VVCase TimingAnalysis. Moreover, the attribute Text of the VVActualOutcome is set to the URLs of the folders containing the generated RCM models and their analysis results. At this point, the engineer can select any of the 16 RCM models satisfying the specified age and reaction constraints and continue with the synthesis of the code for the target platform.

| RCM Model ID	Calculated Age Delay (microseconds)	Calculated Reaction Delay (microseconds)
1	21630	31170
2	21380	30920
3	21480	31020
4	21380	30920
5	21730	31270
6	21380	30920
7	21380	30920
8	21380	30920
9	11810	21810
10	11560	21560
11	11660	21660
12	11560	21560
13	11910	21910
14	11560	21560
15	11560	21560
16	11560	21560
17	11690	21690
18	11440	21440
19	11540	21540
20	11440	21440
21	11790	21790
22	11440	21440
23	11440	21440
24	11440	21440
25	21630	31170
26	21380	30920
27	21480	31020
28	21380	30920
29	21730	31270
30	21380	30920
31	21380	30920
32	21380	30920

Specified Age Constraint: 15000 microseconds
Specified Reaction Constraint: 25000 microseconds
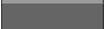
     = satisfied
     = best

TABLE I: Age and Reaction delay analysis results

VI. DISCUSSION AND VALIDATION

In this work, we introduced a novel model-driven methodology, MoVES, for the software development of real-time distributed vehicular embedded systems on single- and multi-core platforms. Considering the importance of timing, the proposed methodology supports the development and architectural exploration of system-designs with temporal awareness. To this end, MoVES leverages the interplay of two domain-specific modelling languages, EAST-ADL and RCM, and provides a fully automated mechanism for the generation of the RCM models containing detailed software, execution platform, timing and allocation information for schedulability analysis. EAST-ADL has been developed by the automotive industry and its consortium includes several international automotive companies such as McLaren, Volvo, FIAT, Hyundai, etc. The validation of the applicability and correctness of EAST-ADL is

outside the scope of this work, however, the interested reader, can refer to [3]. The RCM metamodel definition exploited in this work is an extension of the previous definition given in [12]. Through the years the applicability and correctness of RCM has been verified against several industrial system designs such as i) the Intelligent Parking Assist (IPA) System [25] (consisting of 2 Node elements and 42 SWCs), ii) the simplified IPA system [26] (consisting of 2 Node elements and 7 SWCs), iii) the Steer-by-wire System [20] (consisting of 1 Node and 6 SWCs) and the Brake-by-wire System [12] (consisting of 3 Node elements and 14 SWCs). Additionally, its industrial relevance has been acknowledged by our industrial partners (e.g., Volvo CE, BAE Systems, etc.) through several national and international projects [27], [28].

The automation mechanism, core of the MoVES methodology, is realised by means of a suite of six model transformations. In this respect, the case study presented in Section V helps in discussing some interesting properties of the model transformations, such as syntactic and semantic correctness, complexity, termination and performance. With the term syntactic correctness, we refer to the ability of a transformation to produce valid target models when executed on valid source models [29]. Such a property holds for transformations leveraged in MoVES and the interested reader can easily check the validity of the generated RCM models by accessing the MoVES implementation[11] With the term semantic correctness, we refer to the ability of a transformation to produce semantically valid target models [29]. In this respect, it is important to note that none of the transformations within MoVES suffer of information loss. That is, they consist of a precise and finite set of rules for mapping EAST-ADL to RCM elements without altering, violating or colliding the structural hierarchies of the languages. Eventually, the semantic of the generated RCM models was validated by the leveraged schedulability analysis. We consider two dimensions for the transformations' complexity. The first dimension of the transformations' complexity refers to the complexity of the generated RCM models. In this respect, it is important to note that the transformations always generate RCM models of equal complexity of manually defined RCM models. The second dimension of the transformations' complexity refers to the size of the generated set of RCM models. All the leveraged model transformations, except FDA2RCM and ALLOCATION, are one-to-one. For the FDA2RCM transformation, if we set n as the number of the software functions enclosed by the timing constraints, then FDA2RCM would generate a maximum of $2^{(n-1)}$ RCM models. For instance, in the case of the ACC feature, despite the EAST-ADL FunctionalDesignArchitecture models 9 software functions, the FDA2RCM generates only 4 RCM models as the specified age and reaction constraints enclose 3 software functions. It is important to remark that, as discussed in Section V, logic constraints can be applied for reducing the number of the generated RCM models by discarding those models which entails configurations of timing and control elements that are known to be not relevant. For the ALLOCATION transformation, if we set k as the number of RCM Allocator elements and n as the number of the RCM Allocated elements, then ALLOCATION would generate a maximum of $k^{(n)}$ RCM models. For instance, in

---

[11] http://www.mrtc.mdh.se/MoVES/

the case of the ACC feature, we decided to proceed with an allocation strategy assigning the $n = 3$ RCM Assembly to the $k = 2$ Partition elements. Accordingly, a total number of 8 RCM models containing unique allocation configurations was generated. Similar to FDA2RCM, logic constraints can be specified for narrowing the generation process. For instance, let us consider the case in which we are not interested in the RCM models containing allocation configuration where the EngineControlModule Assembly is not allocated to the Partition_2. In other words, we fix the allocation of the EngineControlModule Assembly to the Partition_2. In this case, ALLOCATION would generate only 4 RCM models. Trivially, the transformations' complexity can affect the termination and performance properties. Although providing formal proof on these properties was outside the scope of the work, the case study showed that all the transformation terminate in few seconds[12].

We believe that MoVES discloses the opportunity to improve the cost-efficiency of software development process by means of i) automation and ii) reduced need for late modifications on the software. In particular, automation by means of model transformations allows to cut the development time while ensuring the compliance with the non functional requirements of the vehicular embedded software. Without MoVES, in fact, the software development would progress incrementally with team of engineers manually defining implementation models until a suitable one, from a non functional perspective, is found. On the contrary, by adopting MoVES, the implementation models would be automatically generated and non functional requirements verified at once allowing the engineers to focus and reason only on the compliant ones. Kurt-Lennart Lundbäck, CEO of Arcticus Systems, on the use of MoVES:

"I feel that autonomous vehicles on multi-core platforms are introducing a lot more of complexity and concerns. In this domain, automation can be a game-changer. For us, as a tool and technology providers, it would be particularly beneficial to have automated support for things like 'allocation' for reducing the complexity of our tool suite, Rubus ICE, and improve its usability. For our customers, this can result in lower development effort and improved confidence in the quality of the software under development."

It is important to note that MoVES does not introduce accidental complexity in the software development process. In fact, it is true that setting up the methodology might require an additional effort, but it is a one-time-effort as opposed to manual processes always requiring constant effort. Moreover, the engineer would have to deal only with the set of RCM models satisfying the non functional requirements which, as shown in Section V for timing, is a limited number (5 out of the 32 generated RCM models). By allowing early verification, MoVES discloses the opportunity of reducing late modifications on the vehicular embedded software, which empirical studies showed to be up to 40 times more expensive than same modifications during the design of the software [6]. In fact, by using MoVES, the engineer is either notified on the non compliance of the starting EAST-ADL models to the set of the considered non functional requirements or notified with

the set of the compliant RCM models with which proceed for the development. In the former scenario, late modifications are prevented while in the latter they are not needed.

In this article, MoVES is presented by specifically targeting timing properties, given the paramount relevance of these properties in the design and development of vehicular embedded systems. Nonetheless, there are further non functional properties that play an important role during the development of these systems, namely memory usage, energy efficiency, and so forth. In this respect, it is worth to note that the methodology proposed by MoVES can be instantiated to consider these and other properties, as long as they are measurable and comparable at the EAST-ADL and RCM levels of detail. Additionally, other properties can be exploited for comparing multiple RCM models having equally good timing performance and selecting the best available RCM model solution. Moreover, further non functional properties could be considered from the initial stages of the proposed workflow to be integrated and used during the generation process of the possible solutions. In both cases, the MoVES would need to be extended only in terms of specific model transformations for the generation of the related non functional properties of interest.

VII. RELATED WORK

This article deals with several research problems, here grouped as development of vehicular software systems, development of multi-core systems, and support for design-space exploration. In the remainder of this section, for each of the mentioned problems relevant related works are discussed.

A. Development of vehicular embedded systems

The growing complexity of nowadays vehicular software demands adequate approaches for its effective development. AUTOSAR [4] is an industrial initiative to provide a standardised software architecture for the development of vehicular software systems. The timing model for AUTOSAR was developed in the TIMMO and TIMMO2USE projects [30], [31]. In these projects, a framework was developed to specify the end-to-end timing constraints and analyse the corresponding end-to-end delays [32], [33], [15]. In general, AUTOSAR is not meant to be used in isolation, but plays the role of the implementation level as e.g. prescribed by the EAST-ADL. Even if the layered structuring of EAST-ADL entails abstraction and separation-of-concerns in the development, there is no specific automation support for interconnecting the different layers. As a consequence, the results of analysis performed at lower levels of abstraction, e.g. by means of AUTOSAR, have to be manually tracked back to higher levels, e.g. design models. More in general, the discontinuities in the development process due to the abstraction gaps between the different layers have to be tackled manually by the developer. This task can be time-consuming and error-prone, especially when considering the complexity of modern vehicular systems [34]. On the contrary, in this article we propose to leverage automation through model transformations to keep the consistency between the different abstraction levels. The abstraction gaps naturally introduce non-determinism, which is managed by an appropriate transformation language (JTL). Multi-core architectures are part of the AUTOSAR standard since version 4.0. However, AUTOSAR does not distinguish between the control and the

---

[12]The case study was run on a 1,7 GHz Intel Core i7 processor, with 8 GB 1600 MHz DDR3 memory.

data flows at the application software level, a distinction that is fundamental for providing early timing verification [35]. Moreover, AUTOSAR does not provide means for modelling the execution platform [36]. These issues motivate our choice of using RCM as the implementation level for EAST-ADL and the MoVES methodology. CHESS is a cross-domain framework for the design of component-based embedded systems, including vehicular systems [37]. It is based on a combination of different languages, like MARTE and SYSML, which gave birth to a specific UML profile. The framework provides modelling of embedded software for early analysis, such as dependability and schedulability, as well as for code generation, monitoring, and back-propagation. Currently, the CHESS framework does not provide design-space exploration, nor it supports uncertainty in the development process. Vehicular systems are often referred to as cyber-physical systems (CPS) [38], especially when considering autonomous driving and networks of vehicles (fleets). Several approaches deal with CPS development by adopting multi-paradigm modelling techniques and leverage simulation mechanisms to perform early analysis of systems [39], [40]. Even if the analyses presented in this article do not exploit simulation techniques, the MoVES methodology does not prevent the use of simulation mechanisms to analyse and select the generated design alternatives with respect to quality attributes of interest.

B. Development of embedded systems

Given the ubiquity of software and its mission criticality, there exists a corpus of literature devoted to the design of embedded systems pertaining to disparate application domains and posing a special focus to QoS requirements. In this respect, several works are based on the use of UML and the UML profile for MARTE [41]. These general-purpose languages might be used as alternatives to domain-specific (i.e. vehicular) languages as, e.g., AUTOSAR and RCM. GASPARD is a framework based on MARTE for the design of parallel embedded systems [42]. It provides a modelling support based on UML and MARTE, and prescribes a workflow made-up of subsequent analyses and refinement steps, from higher to lower abstraction levels. Similarly to MoVES, some analyses and refinements can be performed at the (EAST-ADL) design level, while others can only be performed at lower abstraction levels (e.g., timing). Also for GASPARD moving from higher to lower abstraction levels raises the issue of managing multiple alternatives. Indeed, in [42] the authors advocate for a refinement process able to prune inadequate alternatives based on analysis results. However, the authors do not discuss the management of multiple alternatives, nor provide details about the refinement process that seems to rely on the selection of a single candidate for each level of abstraction. VERTAF/Multi-core [43] is a UML-based framework for the development of multi-core software. The software system is described by means of UML class diagrams, timed state machines and sequence diagrams, while model transformations are used for enabling analysis like schedulability. MARTE is adopted also in [44] and [45] to design the high-level architecture of the software system and for the generation of implementation code. The former approach prescribes the use of UML for modelling the software components while MARTE is used for modelling hardware and software to hardware allocations. Moreover, timing verification is accomplished by running simulations on the automatically generated code. The latter approach instead targets component-based system deployment. Components allocations are derived by means of code generation, which is based on high level description models conforming to MARTE. In [46], the authors propose a technique to specify tasks and their allocation to cores. The technique is based MARTE and allows to perform simulation and task allocation optimisation based on the execution. AADL [47] is an architecture description language initially tailored to the avionic domain but currently used for modelling embedded systems in general. AADL supports the design of multi-core embedded software by means of separation of concerns between software and hardware elements. The software architecture is described in terms of, e.g., Processes and Threads, that is at a lower level of abstraction if compared to RCM.

C. Support for design-space exploration

Design-space exploration (DSE) typically involves the generation, analysis, and optimisation of multiple design alternatives [48]. The step-by-step expansion of design alternatives illustrated in this article can be classified as rule-based DSE complying to the model generation pattern. In fact, the space of solutions is represented by means of models and the corresponding alternatives are generated through model transformations [49], [50]. Moreover, an exhaustive derivation of models at the implementation level is performed by enriching design level (EAST-ADL) models with timing and allocation details, constrained by the system architecture and domain-specific rules [51]. Such a derivation process is quality-driven [49], [52], in the sense that JTL model transformations generate all the viable (timing/allocation) solutions for a certain system architecture, while do not aim at automatically discovering optimisation opportunities at design level [53]. Kang et al. introduce in [54] a DSE tool called FORMULA. FORMULA permits the user to define equivalence classes over alternatives, such that only non-isomorphic solutions are generated in the exploration phase. Their aim is to enhance the cost-effectiveness of DSE by avoiding the exploration of design alternatives not relevant at a certain development stage. Indeed, some alternatives might look equivalent to the user whom, due to the maturity of the design, might not yet be concerned with some of the details about the system that are changed. Several DSE mechanisms cope with the search of input model configurations such that to achieve an optimised system in terms of certain properties of interest. Since the search space is typically huge, research efforts are devoted to generating candidates in an effective way (e.g., close to the optimal solution). Optionally, user??s inputs and/or heuristics are exploited to drive the exploration and prune alternatives. DESERT [55] is a tool that provides DSE based on exploration and pruning rules manually defined by the user. Interestingly, also DESERT is based on a compact representation of viable design alternatives, in particular by means of ordered binary decision diagrams. Differently to the approach proposed by MoVES, DESERT forces the user to perform an element-by-element selection of the available options to reduce the space of solutions to a single one. Shaetz et al. [49] proposed a rule-based DSE mechanism tailored to embedded systems development. The exploration is realised by means of model transformations specified as Prolog programs. In particular,

transformation rules both define the generation of alternatives and constrain the space of solutions. However, there is no explicit visualisation technique to reveal available alternatives to the user. Therefore, the user has to foresee possible design alternatives and write appropriate exploration/pruning rules. A number of additional techniques target multi-criteria optimisations. In this respect, one precondition to be met by the DSE mechanism is a generic representation of the solution space, which has to be compatible with multiple exploration approaches. In general these techniques leverage intermediate formats over which several explorations/optimisations can be run. Notably, in [56] Saxena and Karsai introduce an extension of a domain-specific language devoted to DSE. Such extension is exploited by disparate constraint solvers to compute multiple explorations/optimisations. Similarly, Octopus [57] supports DSE for software intensive embedded systems. A DSE tailored intermediate representation is exploited to implement an iterative refinement process based on analyses, searches, and diagnostics over the space of available solutions. In MoVES, the uncertainty points can be combined to address multiple DSE needs. Moreover, the order of resolutions (i.e., solution exploration) can be exploited to set priorities over properties.

During the last decades, several approaches for the effective software development of vehicular embedded systems were introduced. In general, this problem requires the consideration of a number of models which can rapidly become unbearable to handle manually. In this work, we tacked this problem by employing i) the interplay of two modelling languages, ii) a fully automated mechanism and iii) model-based timing analysis. While the interplay of the two modelling languages allows for the explicit modelling of the vehicular's software architecture and its execution platform, the automated mechanism provides for the automatic generation of all the model alternatives meaningful from a timing perspective.

VIII. Conclusion and Future Work

The work presented in this paper describes a timing-aware model-driven methodology for the software development of distributed vehicular embedded systems on single- and multi-core. In particular, it tackles the problem of guiding the engineer in taking timing-aware design decisions at design level when modifications on the vehicular embedded system are generally more cheaper than modifications at the implementation level. Generally, this requires the consideration of a number of implementation alternatives which are hard to handle without an automated support. We proposed to solve this by introducing automation in terms of six model-transformations which describe precise relationships between EAST-ADL (the language used at design level) and RCM (the language used at implementation level). We exploited the properties of a constraint-based transformation language, JTL, to automatically derive all the possible RCM models entailing meaningful and unique timing and allocation configurations. Eventually, we leveraged model-based timing analysis for the timing verification of the generated RCM models. The case study we conducted together with our industrial partners in the automotive domain demonstrated i) promising results in terms of reduction of late modifications as well as the i) applicability of the methodology.

Despite the generation of the RCM models entailing differ-

ent timing and allocation configurations is transparent to the engineer and it can be guided with logic constraints, issues about scalability and performance may remain open. In this respect, the main future investigation direction encompasses the study of a smarter generation process for narrowing and clustering the space of the generated RCM models and the use of further non functional properties for pruning the set of the generated RCM models. In addition, we are planning to equip the proposed methodology with the compact notation discussed in [7] for enabling the visualisation of multiple RCM models as a single RCM model equipped with uncertainty points. Another line of future investigation encompasses the extension of the proposed methodology to the higher EAST-ADL structural abstraction levels.

References

[1] R. N. Charette, "This car runs on code," *IEEE Spectrum*, vol. 46, no. 3, p. 3, 2009.

[2] D. C. Schmidt, "Guest Editor's Introduction: Model-Driven Engineering," *Computer*, vol. 39, no. 2, pp. 25–31, Feb. 2006.

[3] "EAST-ADL Domain Model Specification, Deliverable D4.1.1, 2010," http://www.atesst.org/home/liblocal/docs/ATESST2_D4.1.1_EAST-ADL2-Specification_2010-06-02.pdf.

[4] "AUTOSAR Techincal Overview, Version 4.3, The AUTOSAR Consortium, Dec., 2016," http://autosar.org.

[5] K. Hänninen, J. Mäki-Turja, M. Sjödin, M. Lindberg, J. Lundbäck, and K.-L. Lundbäck, "The rubus component model for resource constrained real-time systems," in *3rd IEEE International Symposium on Industrial Embedded Systems*, June 2008.

[6] D. Galin, *Software quality assurance: from theory to implementation.* Pearson Education India, 2004.

[7] R. Eramo, A. Pierantonio, and G. Rosa, "Managing uncertainty in bidirectional model transformations," in *Proceedings of the 2015 ACM SIGPLAN International Conference on Software Language Engineering.* ACM, 2015, pp. 49–58.

[8] T. Zan, H. Pacheco, and Z. Hu, "Writing bidirectional model transformations as intentional updates," in *Companion Proceedings of the 36th International Conference on Software Engineering.* ACM, 2014, pp. 488–491.

[9] A. Cicchetti, D. Di Ruscio, R. Eramo, and A. Pierantonio, "Jtl: A bidirectional and change propagating transformation language," in *Software Language Engineering*, 2011, vol. 6563, pp. 183–202.

[10] S. Sendall and W. Kozaczynski, "Model transformation: The heart and soul of model-driven software development," *Software, IEEE*, vol. 20, no. 5, pp. 42–45, 2003. [Online]. Available: http://dx.doi.org/10.1109/MS.2003.1231150

[11] B. Graaf, M. Lormans, and H. Toetenel, "Embedded software engineering: the state of the practice," *Software, IEEE*, vol. 20, no. 6, pp. 61–69, 2003.

[12] A. Bucaioni, S. Mubeen, F. Ciccozzi, A. Cicchetti, and M. Sjödin, "Technology-preserving transition from single-core to multi-core in modelling vehicular systems," in *13th European Conference on Modelling Foundations and Applications*, Springer, Ed., July 2017. [Online]. Available: http://www.es.mdh.se/publications/4750-

[13] ISO 26262-1:2011: Road vehicles in Functional safety. http://www.iso.org/.

[14] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocessing and microprogramming*, vol. 40, no. 2, pp. 117–134, 1994.

[15] 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 *Proceedings of the IEEE Real-Time System Symposium ? Workshop on Compositional Theory and Technology for Real-Time Embedded Systems,*, 2008.

[16] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "Communications-Oriented Development of Component- Based Vehicular Distributed Real-Time Embedded Systems," *Journal of Systems Architecture*, vol. 60, no. 2, pp. 207–220, 2014.

[17] 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," in *Computer Science and Information Systems, vol. 10, no. 1, pp 453-482, January 2013*.

[18] A. Bucaioni, A. Cicchetti, and M. Sjödin, "Towards a metamodel for the rubus component model," in *1st International Workshop on Model-Driven Engineering for Component-Based Software Systems*, September 2014. [Online]. Available: http://www.es.mdh.se/publications/3676-

[19] A. Bucaioni, A. Cicchetti, F. Ciccozzi, S. Mubeen, and M. Sjödin, "A metamodel for the rubus component model: Extensions for timing and model transformation from east-adl," *Journal of IEEE Access*, vol. 5, no. 1, December 2016. [Online]. Available: http://www.es.mdh.se/publications/4611-

[20] 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 *International Workshop on Modelling in Automotive Software Engineering*, September 2015. [Online]. Available: http://www.es.mdh.se/publications/4022-

[21] M. Gelfond and V. Lifschitz, "The stable model semantics for logic programming." vol. 88. MIT Press, 1988, pp. 1070–1080.

[22] "Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT)." OMG Group. [Online]. Available: {http://www.omg.org/spec/QVT/}

[23] K. Czarnecki and S. Helsen, "Feature-based survey of model transformation approaches," *IBM Systems Journal*, vol. 45, no. 3, pp. 621–645, 2006.

[24] P. Berggren, "Autonomous cruise control for chalmers vehicle simulator," 2008.

[25] 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 *International Conference on Information Technology: New Generations*. IEEE, April 2014. [Online]. Available: http://www.es.mdh.se/publications/3468-

[26] 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 *42nd Euromicro Conference series on Software Engineering and Advanced Applications*, September 2016. [Online]. Available: http://www.es.mdh.se/publications/4362-

[27] S. Mubeen, H. Lawson, J. Lundbäck, M. Gålnander, and K. L. Lundbäck, "Provisioning of predictable embedded software in the vehicle industry: The rubus approach," in *2017 IEEE/ACM 4th International Workshop on Software Engineering Research and Industrial Practice (SER IP)*, May 2017, pp. 3–9.

[28] "Arcticus Systems' Research Projects," http://www.arcticus-systems.com/research/.

[29] T. Mens and P. Van Gorp, "A taxonomy of model transformation," *Electronic Notes in Theoretical Computer Science*, vol. 152, pp. 125–142, 2006.

[30] "TIMMO-2-USE," https://itea3.org/project/timmo-2-use.html.

[31] "Mastering Timing Information for Advanced Automotive Systems Engineering. In the TIMMO-2-USE Brochure, 2012. Available at: http://www.timmo-2-use.org/pdf/T2UBrochure.pdf," 2012.

[32] F. Stappert, J. Jonsson, M. Jürgen, and J. Rolf, "A Design Framework for End-To-End Timing Constrained Automotive Applications," in *Embedded Real-Time Software and Systems (ERTS)*, 2010.

[33] Timing Augmented Description Language (TADL2) syntax, semantics, metamodel Ver. 2, Deliverable 11, Aug. 2012.

[34] B. Selic and L. Motus, "Using models in real-time software design," *Control Systems, IEEE*, vol. 23, no. 3, pp. 31–42, June 2003.

[35] S. Mubeen, T. Nolte, M. Sjödin, J. Lundbäck, and K.-L. Lundbäck, "Supporting timing analysis of vehicular embedded systems through the refinement of timing constraints," *Software & Systems Modeling*, Jan 2017. [Online]. Available: https://doi.org/10.1007/s10270-017-0579-8

[36] A. Sangiovanni-Vincentelli and M. Di Natale, "Embedded system design for automotive applications," *Computer*, vol. 40, no. 10, pp. 42–51, Oct. 2007.

[37] A. Cicchetti, F. Ciccozzi, S. Mazzini, S. Puri, M. Panunzio, T. Vardanega, and A. Zovi, "Chess: a model-driven engineering tool environment for aiding the development of complex industrial systems," in *27th International Conference on Automated Software Engineering (ASE 2012)*, September 2012. [Online]. Available: http://www.es.mdh.se/publications/2208-

[38] P. Derler, E. A. Lee, and A. Sangiovanni-Vincentelli, "Modeling cyber-physical systems," *Proceedings of the IEEE (special issue on CPS)*, vol. 100, no. 1, pp. 13 – 28, January 2012. [Online]. Available: http://chess.eecs.berkeley.edu/pubs/843.html

[39] P. J. Mosterman and H. Vangheluwe, "Computer automated multi-paradigm modeling: An introduction," *SIMULATION*, vol. 80, no. 9, pp. 433–450, 2004. [Online]. Available: http://dx.doi.org/10.1177/0037549704050532,

[40] J. C. Jensen, D. H. Chang, and E. A. Lee, "A model-based design methodology for cyber-physical systems," in *2011 7th International Wireless Communications and Mobile Computing Conference*, July 2011, pp. 1666–1671.

[41] "The UML Profile for MARTE: Modeling and Analysis of Real-Time and Embedded Systems, 2010." OMG Group, January 2010. [Online]. Available: {http://www.omgmarte.org/}

[42] A. Gamatié, S. Le Beux, É. Piel, R. Ben Atitallah, A. Etien, P. Marquet, and J.-L. Dekeyser, "A model-driven design framework for massively parallel embedded systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 10, no. 4, p. 39, 2011.

[43] P.-A. Hsiung, S.-W. Lin, Y.-R. Chen, N.-L. Hsueh, C.-H. Chang, C.-H. Shih, C.-S. Koong, C.-S. Lin, C.-H. Lu, S.-Y. Tong, W.-T. Su, and W. C. Chu, "Model-driven development of multi-core embedded software," in *Proceedings of the 2009 ICSE Workshop on Multicore Software Engineering*, ser. IWMSE '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 9–16.

[44] F. Ciccozzi, T. Seceleanu, D. Corcoran, and D. Scholle, "Uml-based development of embedded real-time software on multi-core in practice: Lessons learned and future perspectives," *Journal of IEEE Access*, vol. 2, no. 1, pp. 1–12, September 2016. [Online]. Available: http://www.es.mdh.se/publications/4471-

[45] A. Nicolas, H. Posadas, P. Peñil, and E. Villar, "Automatic deployment of component-based embedded systems from uml/marte models using mcapi," in *Design of Circuits and Integrated Systems*, Nov 2014, pp. 1–6.

[46] F. Ciccozzi, J. Feljan, J. Carlson, and I. Crnkovic, "Architecture optimization: Speed or accuracy? both!" *Software Quality Journal*, vol. 22, no. 1, pp. 1–28, October 2016. [Online]. Available: http://www.es.mdh.se/publications/4521-

[47] P. H. Feiler, D. P. Gluch, and J. J. Hudak, "The architecture analysis & design language (aadl): An introduction," DTIC Document, Tech. Rep., 2006.

[48] M. Gries, "Methods for evaluating and covering the design space during early design development," *Integr. VLSI J.*, vol. 38, no. 2, pp. 131–183, Dec. 2004.

[49] B. Schätz, F. Hölzl, and T. Lundkvist, "Design-space exploration through constraint-based model-transformation," in *Engineering of Computer Based Systems (ECBS), 2010 17th IEEE International Conference and Workshops on*, March 2010, pp. 173–182.

[50] H. Abdeen, D. Varró, H. Sahraoui, A. S. Nagy, C. Debreceni, A. Hegedüs, and A. Horváth, "Multi-objective optimization in rule-based design space exploration," in *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ser. ASE '14. New York, NY, USA: ACM, 2014, pp. 289–300.

[51] K. Vanherpen, J. Denil, P. De Meulenaere, and H. Vangheluwe, "Design-space exploration in model driven engineering," SOCS-TR-2014.4, McGill University, Tech. Rep., 2014.

[52] M. L. Drago, C. Ghezzi, and R. Mirandola, "Towards quality driven exploration of model transformation spaces," in *Procs. of the 14th Int. Conf. on Model Driven Engineering Languages and Systems*, ser.

MODELS'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 2–16. [Online]. Available: http://dl.acm.org/citation.cfm?id=2050655.2050659

[53] M. Walker, M.-O. Reiser, S. Tucci-Piergiovanni, Y. Papadopoulos, H. Lnn, C. Mraidha, D. Parker, D. Chen, and D. Servat, "Automatic optimisation of system architectures using east-adl," *Journal of Systems and Software*, vol. 86, no. 10, pp. 2467–2487, 2013.

[54] E. Kang, E. Jackson, and W. Schulte, *Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems: 16th Monterey Workshop 2010, Redmond, WA, USA, March 31- April 2, 2010, Revised Selected Papers*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, ch. An Approach for Effective Design Space Exploration, pp. 33–54.

[55] S. Neema, J. Sztipanovits, G. Karsai, and K. Butts, *Embedded Software: Third International Conference, EMSOFT 2003, Philadelphia, PA, USA,* *October 13-15, 2003. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, ch. Constraint-Based Design-Space Exploration and Model Synthesis, pp. 290–305.

[56] T. Saxena and G. Karsai, *Model Driven Engineering Languages and Systems: 13th International Conference, MODELS 2010, Oslo, Norway, October 3-8, 2010, Proceedings, Part I*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, ch. MDE-Based Approach for Generalizing Design Space Exploration, pp. 46–60.

[57] T. Basten, M. Hendriks, N. Trčka, L. Somers, M. Geilen, Y. Yang, G. Igna, S. Smet, M. Voorhoeve, W. Aalst, H. Corporaal, and F. Vaandrager, *Model-Based Design of Adaptive Embedded Systems*. New York, NY: Springer New York, 2013, ch. Model-Driven Design-Space Exploration for Software-Intensive Embedded Systems, pp. 189–244.