Maximilian Eibl, Martin Gaedke. (Hrsg.): INFORMATIK 2017,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2017 1487

Automotive Systems Modelling with Vitruvius

Manar Mazkatli[1] Erik Burger[1] Anne Koziolek[1] Ralf H. Reussner[1]

Abstract: Model-driven technologies are widely used in the development of systems in the automotive domain. Although modelling tools and code generation increase the development speed and the quality of the developed system in general, the availability of several modelling languages for different stages of the development process also introduces additional problems: Developers use several models to describe the same system on different levels of abstraction, which serve as documentation, basis for code generation, but also for model-based analyses of system properties such as security, performance, or reliability. If models are modified independently, inconsistencies can arise, which lead to incorrect results of these analyses, complicate the implementation of new features, and create errors at later stages in development that are costly to fix.

In this paper, we apply the model-based Vitruvius approach, which preserves consistency in heterogeneous modelling environments, to a scenario of automotive systems development. The scenario includes the modelling standards SysML, AMALTHEA and ASCET. We show, at the example of an onboard controlling unit, how the Vitruvius approach can be used to increase consistency in automotive system development and reduce the accidental complexity that arises for developers who have to work with heterogeneous modelling languages.

Keywords: Vitruvius, view-based development process, model-based development process, automotive systems, declarative description of correspondence rules.

1 Introduction

The development of a technical system includes the modelling of the system from different perspectives and at different levels of abstraction. The models that are created during this process may be models of hardware, software, and additional information such as non-functional properties. Program code in general-purpose languages such as C is then often generated from these models.

In the automotive domain, for example, the SysML and AUTOSAR standards are widely used in conjunction with specific platforms such as AMALTHEA or ASCET. Often, there is no specified interface for the exchange of models between these standards. Thus, developers exchange the needed information and documentation along the development process and are often forced to manually check for consistency between the models. This process is,

[1] Karlsruhe Institute of Technology, Institute for Program Structures and Data Organisation (IPD)
Am Fasanengarten 5, 76131 Karlsruhe, Germany
{manar.mazkatli|burger|koziolek|reussner}@kit.edu

doi:10.18420/in2017_148

however, expensive because a great part of the exchanged files is written by hand, and the reuse of information is done using error-prone techniques, e.g., copy and paste.

During the evolution of a system, developers may also alter these models independently of each other. A modification in one of the models can lead to an inconsistent description of the system if it is not manually propagated to all other models, which may in turn lead to inaccurate simulations and analyses, for example security or performance analyses. If inconsistencies affect the generated program code negatively, this problem is only be detected in late stages of the development process, e.g., during the assembly phase. Since compilation of program code can take up several hours in systems of common size, the correction of these inconsistency errors is very time-consuming.

VITRUVIUS [KBL13; Bur14] is a model-based approach for the development of software with heterogeneous modelling standards. It contains declarative languages for consistency relations between heterogeneous models that describe the same system, which are used to detect violations and to re-establish consistency in the system model. In this paper, we will describe how we have applied the VITRUVIUS approach to automotive systems engineering to improve the reliability of automotive systems. We have conducted a case study that describes the development of an onboard control unit for automotive systems, using the languages and standards SysML, AMALTHEA, and ASCET. To use these languages in the *Eclipse Modeling Framework*, which is the technical space of the VITRUVIUS prototype, we have selected appropriate metamodels and defined an import procedure for existing models. To express the consistency relations between these types of models, we have defined declarative mapping specifications in the Mappings language of VITRUVIUS.

After a description of the foundations of model-driven engineering and especially the VITRUVIUS approach in section 2, we will introduce our case study of an automotive software controller in section 3. In section 4, we describe the application of VITRUVIUS in our case study. After related work (section 5) and future work (section 6), we conclude the paper with section 7.

2 Foundations

2.1 Model-Driven Engineering

Model-Driven engineering is a development paradigm that puts models in the centre of the development process and uses the following concepts [Sch06]: First, *domain-specific languages (DSL)* express domain concepts, which reduce the complexity in the modelling of systems. DSLs are defined using *metamodels*, which themselves are defined using a standardised, fixed meta-metamodel. Second, *transformations engines* are used to transform these models into instances of other DSLs or into textual representations, such as code.

The Eclipse Modeling Framework (EMF)[2] is a development framework for model-driven development that is implemented using the Java-based Eclipse platform.

2.2 Vitruvius

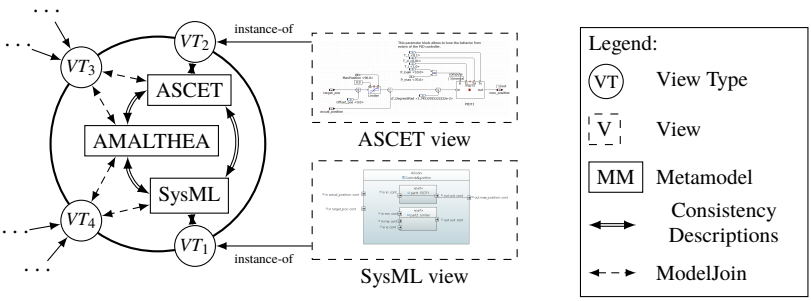

Fig. 1: The modular SUM Metamodel concept of Vitruvius at the example of automotive systems engineering

Vitruvius [KBL13] is a view-based, model-driven framework for the management of heterogeneous models, i.e., models that are instances of different metamodels. It is based on the concept of a *single underlying model (SUM)* [ASB10], which represents all the information that is available about the system under development, and implements this concept into a *virtual SUM (VSUM)* that encapsulates existing metamodels and enriches them with correspondence information and custom *views*, which are specialized models, following the definition of Goldschmidt et al. [GBB12]. The VSUM conforms to a customized metamodel that is specific to the domain in which the Vitruvius approach is used; for example, in the automotive domain, it may contain the metamodels of SysML, AMALTHEA and other standards, which are combined to form a modular SUM metamodel (see Figure 1). The metamodels are included non-intrusively and do not have to be adapted. To express the semantic relations between the elements of the metamodels, Vitruvius defines a language framework for consistency description and restoration that consists of three languages for *reactions*, *mappings* and *invariants*. Since Vitruvius is a view-based approach, all information in the SUM can only be retrieved or manipulated via specialized views. For the definition of view types and views, Vitruvius uses the *ModelJoin* language [Bur+14]. The consistency preservation mechanism is triggered by changes to one or several views. The preservation mechanism of Vitruvius then reacts on a list of changes to propagate them to the SUM.

Vitruvius has been implemented as a prototype[3] in the Eclipse Modeling Framework and can thus be used with any Ecore-conforming metamodel. So far, it has been applied to software architecture models and model-based representations of programming languages

[2] `http://www.eclipse.org/modeling/emf/`, retrieved 2016-12-13

[3] `https://sdqweb.ipd.kit.edu/wiki/Vitruvius`, retrieved 2017-02-13

[LK15]. It has been also applied to component-based software development [Kra+15] and systems modelling of energy networks [BMK16].

Leonhardt et al. [Leo+15] have introduced two strategies to integrate one legacy model into VITRUVIUS. Moreover, we have extended VITRUVIUS concept in [Maz16] to check the consistency between several legacy models automatically, resolve the conflicts semi-automatically (if founded) and integrate these models in VITRUVIUS platform.

3 Case Study of PID Controller Software Development

This case study describes the development of a controller software using a traditional PID (Proportional, Integral, and Derivative) control algorithm. This algorithm is commonly used in the automotive field (e.g., for controlling throttle positions). It depends on a control loop feedback mechanism to calculate an error value as the difference between a desired set point (`target_position` in our case study) and a measured process variable (`actual_position` in our case study). Then it attempts to minimize this error over time by adjustment of a control variable (`new_position` in our case study). Figure 2 shows the layout of the Electronics Control Unit (ECU) that will later execute the control algorithm. In this development scenario, the developers describe the structure of the controller software using *SysML*. Then they describe the software architecture in *AMALTHEA* platform to generate glue-code describing tasks, operating system configuration and the scheduling. They implement the defined architecture using *ASCET* and generate the implementation code. The integration of both codes on circuit provides the final executable code. The following subsections give an overview of the modelling tools and the problem that arises during the development process.

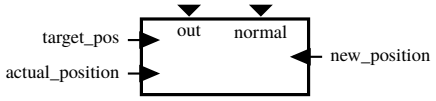

Fig. 2: Layout of the control algorithm ECU

3.1 Languages and Models

SysML (Systems Modelling Language) is a graphical modelling language developed for systems engineering [OMG12]. SysML can depict a system's structure using Block Definition Diagrams (BDD) and Internal Block Diagrams (IBD). BDDs provide a black box representation of a system's blocks and the interconnections between them in term of *flow ports*, whereas IBDs instantiate them to represent the final assembly of the system.

In our case study, we describe the controller software structure using an IBD. Figure 3 shows ControlAlgorithm Block with its internal blocks (PID Tuning Block and Limiter Block) in addition to its in-/out-ports (`actual_position`, `target_position` and `new_position`).

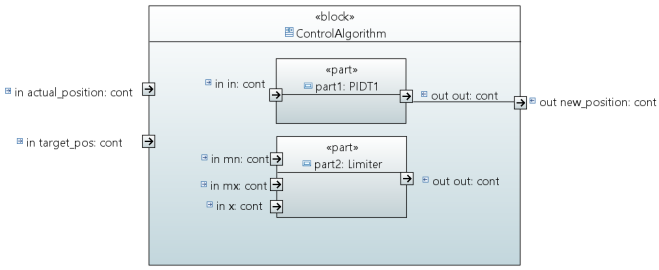

Fig. 3: Modelling the structures of control algorithm case study using IBD

AMALTHEA is an open and expandable tool platform for embedded multicore systems. It combines the developing tools of the automotive ECUs in a single platform [BJ13]. Our case study describes the PID controller software architecture using AMALTHEA component models and software models. Component models define the components (e.g., *ControlAlgorithm*) and the connection between them, whereas the software model defines software units. *Runnables* are executable software units that can be run in parallel (e.g., `ControlAlgorithm_normal` and `ControlAlgorithm_out`). *Labels* are data elements that are located in memory and that are read/written by runnables (e.g., `new_position`, `target_pos`, and `actual_position`). *Processes* are a generalization of tasks that define execution paths calling runnables/other processes (e.g., `_10MS` task). From these models, the developers generate the C code (glue-code) including the OSEK Implementation Language (OIL) files[4] that describe OSEK real time systems (multitasking and communication configuration).

ASCET (The Advanced Simulation and Control Engineering Tool) is a tool suite from ETAS GmbH for model-based development of embedded automotive software. ASCET Modeling and Design (ASCET-MD)[5] is a part of the ASCET product family and offers executable specification of ECU functions using graphical tools (e.g., block diagrams and state machines) or by textual tools (e.g., Embedded Software Description Language (ESDL) editors, and C code editors). For example, the block diagram editor shown in Figure 4 describes the functionality of the ControlAlgorithm software component in terms of an *AscetModule*. The diagram shows its internal components: a pre-defined component *Limiter* provided from the ASCET database, and the *PIDT1* component described in another block diagram. Furthermore, it shows imported data from other blocks, such as messages (`target_pos` and `actual_position`), the output message exported to other blocks (`new_position`), and the flow of the data/control signal through different arithmetical/logical operations. This model generates the C code that will be then integrated with AMALTHEA glue-code.

[4] `http://www.irisa.fr/alf/downloads/puaut/TPNXT/images/oil25.pdf`, retrieved 2016-12-13
[5] `http://www.etas.com/de/products/ascet_md_modeling_design-details.php`, retrieved 2016-12-13

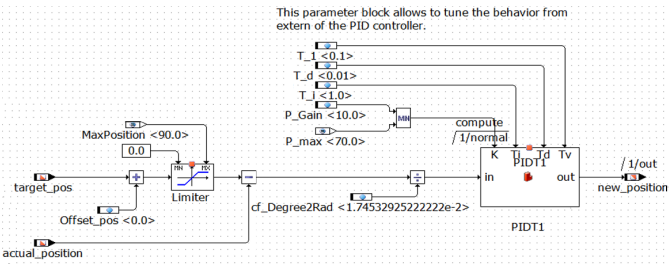

Fig. 4: Modelling of control algorithm in ASCET using block diagram editor

3.2 Consistency Preservation

To keep the consistency between the models in automotive systems domain, the developers usually exchange documents including the needed information and documentation along the development process. Developers may, for example, apply technology based on Manufacturer Supplier Relationship (MSR)[6] [SLG05] and XML [WH04]. This technology stores the information in a uniform format once in a shared database. As explained in section 1, this approach suffers from manual production and synchronisation of documents, which can lead to inconsistencies. This may result in high costs for resolving potential conflicts that are first detected at the assembly stage. Further problems are drift and erosion between models and code, if potential inconsistencies are resolved only by correcting code errors.

4 Preserving Consistency with Vitruvius Approach

This section describes the consistency preservation of our case study based on Vitruvius. First, we explain how we prepare Vitruvius platform to be used for consistent view-based development. Then, we illustrate how we have evaluated the declarative mappings between the metamodels of our case study. Finally, we show the result of the evaluation.

Initializing Vitruvius Platform: In the first step of applying Vitruvius, we create the modular SUM metamodel, which includes an Ecore-based metamodel for each of the three modelling languages SysML, AMALTHEA and ASCET. Since the metamodels of both SysML and AMALTHEA are Ecore-based, we have imported and reused them directly. For ASCET, we have used the ASCET Data Object Model (ADOM) metamodel, which is defined in conjunction with the ASCET XML Model Description (AMD) metamodel.

The three metamodels share a high semantic overlap. Therefore, we have identified the semantic correspondences between them. Table 1 lists the most important correspondences

[6] `http://www.msr-wg.de`, retrieved 2016-12-13

SysML	AMALTHEA	ASCET	Examples from the case study
Block	Component	AscetModule	ControlAlgorithm
FlowPort	Label	Message	target_pos, actual_position, new_position
–	Runnable	Method	normal, out
–	Task	Task, InitTask, etc	_10MS

Tab. 1: Main correspondences between SysML, AMALTHEA and ASCET

with examples from the PID controller case study. To control this redundancy of information according to Vitruvius, we had to define the mapping between the artefacts sharing the same semantic, the invariants that violate the consistency, and the reactions that can restore the consistency when the invariants are violated. In this work, we concentrate on defining the mapping between the metamodels of our case study with declarative mappings language of Vitruvius. List. 1 illustrates an example of the declarative mapping using the Vitruvius mapping language. This example defines the correspondence between the Component metaclass of the AMALTHEA metamodel and the AscetModule metaclass of the ASCET metamodel, which describes the behaviour of this component. The example shows also the sub-mapping and correspondence between Runnable (AMALTHEA) and Method (ASCET) on one hand (lines 6–9) and the AMALTHEA Label and ASCET Message data types on the other hand (lines 10–13). The mapping conditions are written between square brackets. After this, the Vitruvius framework generates the bidirectional model-based transformations that are used for re-instating the consistency.

```
1 import package "http://amalthea.itea2.org/model/1.0/components" as comp
2 import package "http://com.bosch.swan.ascet.adom/1.0" as adom
3 mapping ascetModule_component:
4 map {adom.AscetModule as ascetModule} and {comp.Component as component}
5 {   [equal (ascetModule.(name),component.(name))]
6 map {ascetModule.methods as method} and {component.runnables as runn}
7     {[    equal(Method.name, runn.name)
8           empty(method.arguments)
9           equal(method.ret,null) ]}
10 map {ascetModule.elements as messag} with {messag.type==adom::Message}
11          and component.(labels as label)
12    {[    equal(label. constant,false)
13          equal(label.name,messag.name) ]} }
```

List. 1: Correspondence rule between AscetModule and Component

Evaluation: To evaluate the defined mappings and their implementation (the bidirectional model-based transformation), we wrote a set of test cases that define different scenarios to rebuild the models of the PID controller case study in the Vitruvius platform. According to

each scenario, the test case instantiates the modular SUM metamodel, makes some changes to the models (create, update or delete some objects), and then triggers the synchronisation to propagate them to the related models. Finally, the test case asserts that the changes are correctly propagated. One of the test cases scenario can be the following: creating an AMALTHEA Component object named `ControlAlgorithm`, creating two Runnable objects (`out`, `normal`), and three Label objects (`actual_position`, `target_position`, `new_position`) and assigning them to the ControlAlgorithm component. The expected changes after the synchronisation are: creating the related AscetModule object `ControlAlgorithm` that consists of two Method objects without arguments or return type (`out`, `normal`), and three Message objects (`actual_position`, `target_position`, `new_position`). Moreover, the synchronisation should create a SysML Block instance named `ControlAlgorithm` and containing three FlowPort objects (`actual_position`, `target_position`, `new_position`).

Results: The evaluation of the VITRUVIUS synchronisation based on the defined mapping showed the following issues: First, the VITRUVIUS synchronisation can often re-instate the consistency fully automatically. For example, the test case example defined in the previous section is performed correctly, i.e., the synchronisation automatically creates the related objects (AscetModule and SysML Block objects) with the shared information. Similarly, altering the name of an AMALTHEA Component object will automatically update the name of its related objects. Second, the synchronisation was not able to restore the consistency fully automatically in some cases. For instance, if the developer creates an AMALTHEA Task object named `_10MS`, the synchronisation will ask him then to select the type of ASCET Task object that should be created to restore the consistency. This is due to the one-to-many mapping between the AMALTHEA Task type on one hand and the ASCET Task type and its subtypes on the other hand (see the last row of Table 1). Third, the automatically created related objects may need further development to provide them with the non-shared information. For example, the AscetModule objects that are automatically created by the synchronisation should be further developed to describe the functionality in details. Thus, VITRUVIUS informs the developers with automatically generated elements that may need further development. Finally, the found results show that VITRUVIUS can be used to keep the consistency in a real-life development scenario of automotive heterogeneous models. The main advantages are that the automated synchronisation will save manual effort, avoid potential inconsistency that may occur during the manual process, resolve the conflicts with lower cost in the design stage, and keep the models consistent with their implementation.

5 Related work

This section describes related approaches that are used to keep consistency between automotive heterogeneous models. The document-centric approach (mentioned in section 1 and subsection 3.2) suffers from manual preservation of consistency. Born et al. [BFK10] suggest representing exchanged information as models – even if it is a document – to

ease tracing and to perform semi-automatic checks of consistency. Our approach also offers semi-automatic conflict resolution. Other works specify the relations between the models and use them to perform automatic consistency preservation. For example, Giese et. al. suggest using Triple Graph Grammars (TGGs) [Sch95] to describe the relationship between SysML and AUTOSAR [GHN10], and to generate the bidirectional model-based transformations needed to synchronise these two models with each other. If there are more than two models, additional chains of transformations should be built to connect them. The Vitruvius approach, however, offers the development and management of more than two heterogeneous models. Moreover, any change that violates the consistency constrains will be not processed but rather ignored according to Giese's concept, which is not the case in Vitruvius approach, where changes that violate consistency constraints can be handled by predefined response actions.

Other concepts keep consistency by generating a consistent model from another one using model-based transformations. For example, Selim et al. [Sel+15] apply model transformations to migrate from legacy models of General Motors, which are built using custom-built and domain-specific modelling language, to standardized AUTOSAR models. Model-to-model transformations have been also used by Sindico et al. [SNP11] to generate Simulink models from SysML models and vice versa. Similar work by Sjöstedt et al. [Sjö+08] transforms Simulink models to UML composite structure and activity models based on Atlas Transformation Language ATL. Macher et al. [MAK15] depend on seamless combination of heterogeneous tools approach [Bro+10] to exchange the models between SysML and Matlab/Simulink tools. Other work of Macher [Mac+15] based on the same approach generates the configuration of RTOS from control system information in SysML and vice versa. The approaches mentioned in this paragraph are limited to a specific combination of two models or languages. The Vitruvius approach, in contrast, can be used to combine arbitrary modelling languages, as long as they can be expressed as an Ecore-conforming metamodel. Vierhauser et al. [Vie+12] present a framework for checking and maintaining consistency between the Product Line (PL) model and some parts of underlying code. However, their approach is limited to PL models and their underlying code and does not support other models needed in software product line engineering.

6 Future Work

Currently, editors for the modelling languages used in our case study are not directly usable with Vitruvius. Therefore, we suggest connecting the modelling tools with the Vitruvius platform through implementing bidirectional transmitters. These transmitters should detect the changes made in the modelling tools and transform them to Vitruvius for the synchronisation on one hand. On the other hand, they should record the changes resulting by Vitruvius synchronisation and transform them to the modelling tools in order to update the models under development. If recording the changes in the modelling tool is not feasible, as it is the case of closed-source ETAS ASCET tool, then the transmitter should calculate them by comparing two recent versions of the model in a way similar to [BT14].

7 Conclusion

In this paper, we have demonstrated how the model-driven Vitruvius approach can be used in the development of automotive systems. Following the Vitruvius process, we have described how the existing standards SysML, AMALTHEA and ASCET can be used with the Eclipse-based Vitruvius framework. The case study of an onboard control unit demonstrated that the steps for preserving and re-instating consistency, which are usually carried out manually, can be automated with the Vitruvius framework. We have formulated change-based consistency rules with the declarative Reactions language of Vitruvius to automate steps of consistency preservation that would normally have to be carried out manually and were for this reason often omitted.

It is an important factor for reliable and secure systems that all the artefacts in the development process share consistency, so that they can provide an up-to-date description of the system under development at all stages. Model-driven technologies, which are used widely in automotive development processes, form the basis for the advanced consistency preservation methods of the Vitruvius approach. Thus, the accidental complexity of developing a system with multiple metamodels, languages, or development tools, can be reduced.

References

[ASB10] Colin Atkinson, Dietmar Stoll, and Philipp Bostan. "Orthographic Software Modeling: A Practical Approach to View-Based Development". In: *Evaluation of Novel Approaches to Software Engineering*. Ed. by Leszek A. Maciaszek, César González-Pérez, and Stefan Jablonski. Vol. 69. Communications in Computer and Information Science. Berlin/Heidelberg: Springer, 2010, pp. 206–219.

[BFK10] Marc Born, John Favaro, and Olaf Kath. "Application of ISO DIS 26262 in practice". In: *Proceedings of the 1st Workshop on Critical Automotive applications: Robustness and Safety*. ACM. 2010, pp. 3–6.

[BJ13] Christopher Brink and Jan Jatzkowski. *AMALTHEA (ITEA2 – 09013) – White Paper*. Tech. rep. University of Paderborn, Germany, 2013.

[BMK16] Erik Burger, Victoria Mittelbach, and Anne Koziolek. "Model-driven Consistency Preservation in Cyber-physical Systems". In: *Proceedings of the 11th Workshop on Models@run.time co-located with ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MODELS 2016)*. (Saint Malo, France). CEUR Workshop Proceedings, Oct. 2016.

[Bro+10] Manfred Broy et al. "Seamless model-based development: From isolated tools to integrated model engineering environments". In: *Proceedings of the IEEE* 98.4 (2010), pp. 526–545.

[BT14] Erik Burger and Aleksandar Toshovski. “Difference-based Conformance Checking for Ecore Metamodels”. In: *Proceedings of Modellierung 2014*. Vol. 225. GI-LNI. Vienna, Austria, Mar. 2014.

[Bur+14] Erik Burger et al. “View-Based Model-Driven Software Development with ModelJoin”. In: *Software & Systems Modeling* 15.2 (2014). Ed. by Robert France and Bernhard Rumpe, pp. 472–496.

[Bur14] Erik Burger. “Flexible Views for View-based Model-driven Development”. PhD thesis. Karlsruhe, Germany: Karlsruhe Institute of Technology, July 2014.

[GBB12] Thomas Goldschmidt, Steffen Becker, and Erik Burger. “Towards a Tool-Oriented Taxonomy of View-Based Modelling”. In: *Proceedings of the Modellierung 2012*. Ed. by Elmar J. Sinz and Andy Schürr. Vol. P-201. GI-Edition – Lecture Notes in Informatics (LNI). Bamberg: Gesellschaft für Informatik e.V. (GI), Mar. 2012, pp. 59–74.

[GHN10] Holger Giese, Stephan Hildebrandt, and Stefan Neumann. “Model synchronization at work: keeping SysML and AUTOSAR models consistent”. In: *Graph transformations and model-driven engineering*. Ed. by Gregor Engels et al. Berlin, Heidelberg: Springer, 2010, pp. 555–579.

[KBL13] Max E. Kramer, Erik Burger, and Michael Langhammer. “View-centric engineering with synchronized heterogeneous models”. In: *Proceedings of the 1st Workshop on View-Based, Aspect-Oriented and Orthographic Software Modelling*. VAO ’13. Montpellier, France: ACM, 2013, 5:1–5:6.

[Kra+15] Max E Kramer et al. “Change-Driven Consistency for Component Code, Architectural Models, and Contracts”. In: *Proceedings of the 18th International ACM SIGSOFT Symposium on Component-Based Software Engineering*. ACM. 2015, pp. 21–26.

[Leo+15] Sven Leonhardt et al. “Integration of Existing Software Artifacts into a View- and Change-Driven Development Approach”. In: *Proceedings of the 2015 Joint MORSE/VAO Workshop on Model-Driven Robot Software Engineering and View-based Software-Engineering*. MORSE/VAO ’15. L’Aquila, Italy: ACM, 2015, pp. 17–24.

[LK15] Michael Langhammer and Klaus Krogmann. “A Co-evolution Approach for Source Code and Component-based Architecture Models”. In: *17. Workshop Software-Reengineering und-Evolution*. Vol. 4. 2015.

[Mac+15] Georg Macher et al. “Automotive real-time operating systems: a model-based configuration approach”. In: *ACM SIGBED Review* 11.4 (2015), pp. 67–72.

[MAK15] Georg Macher, Eric Armengaud, and Christian Kreiner. “Integration of Heterogeneous Tools to a Seamless Automotive Toolchain”. In: *Systems, Software and Services Process Improvement*. Ed. by Rory V. O’Connor et al. Cham: Springer International Publishing, 2015, pp. 51–62.

[Maz16] Manar Mazkatli. “Consistency Preservation in the Development Process of Automotive Software”. MA thesis. Karlsruhe Institute of Technology (KIT), 2016.

[OMG12] *OMG Systems Modeling Language (OMG SysML)*. Version 1.3. Object Management Group. June 2012.

[Sch06] D.C. Schmidt. “Guest Editor’s Introduction: Model-Driven Engineering”. In: *Computer* 39.2 (Feb. 2006), pp. 25–31.

[Sch95] Andy Schürr. “Specification of graph translators with triple graph grammars”. In: *Proceedings of the 20th International Workshop on Graph-Theoretic Concepts in Computer Science*. Springer. London, UK, 1995, pp. 151–163.

[Sel+15] Gehan MK Selim et al. “Model transformations for migrating legacy deployment models in the automotive industry”. In: *Software & Systems Modeling* 14.1 (2015), pp. 365–381.

[Sjö+08] Carl-Johan Sjöstedt et al. “Mapping Simulink to UML in the design of embedded systems: Investigating scenarios and transformations”. In: *OMER4 Workshop: 4th Workshop on Object-oriented Modeling of Embedded Real-Time Systems*. 2008, pp. 137–160.

[SLG05] Marek Szwejczewski, Fred Lemke, and Keith Goffin. “Manufacturer-supplier relationships: An empirical study of German manufacturing companies”. In: *International Journal of Operations & Production Management* 25.9 (2005), pp. 875–897.

[SNP11] Andrea Sindico, Marco Di Natale, and Gianpiero Panci. “Integrating SysML with Simulink using Open-source Model Transformations.” In: *SIMULTECH*. Ed. by Janusz Kacprzyk, Nuno Pina, and Joaquim Filipe. SciTePress, 2011, pp. 45–56.

[Vie+12] Michael Vierhauser et al. “Applying a consistency checking framework for heterogeneous models and artifacts in industrial product lines”. In: *International Conference on Model Driven Engineering Languages and Systems*. Springer. 2012, pp. 531–545.

[WH04] Bernhard Weichel and Martin Herrmann. *A backbone in automotive software development based on XML and ASAM/MSR*. Tech. rep. SAE Technical Paper, 2004.