Doctor of Engineering Thesis

Measuring the Evolution of Meta-models, Models and Design Requirements to Facilitate Architectural Updates in Large Software Systems

Darko Đurišić

University of Gothenburg

Division of Software Engineering
Department of Computer Science & Engineering
University of Gothenburg
Gothenburg, Sweden, 2017

Measuring the Evolution of Meta-models, Models and Design Requirements to Facilitate Architectural Updates in Large Software Systems

Darko Đurišić

Technical Report No 148D
ISBN 978-91-982237-5-0

Department of Computer Science & Engineering
Division of Software Engineering
University of Gothenburg
Gothenburg, Sweden

This thesis has been prepared using LaTeX.
Printed by Chalmers Reproservice,
Gothenburg, Sweden 2017.

To my son Filip
Your smiles and laughs came just in time to inspire me to finish up what I started long before you were born.

Abstract

Background: In order to reduce complexity of the system and its development cost, the architecture of large software systems is often developed following the MDE (Model-Driven Engineering) approach. Developing architectures according to MDE relies on three main artifacts in the development process: domain-specific meta-models, architectural models and system design requirements. The architecture of the system is defined in the architectural models which are developed using modeling tools. The syntax of the models is defined in domain-specific meta-models, while their semantics is usually provided in a form of system design requirements in the supporting specifications.

Objective: The main objective of this thesis is to develop methods and tools for managing architectural updates in the development of large software systems. Our goal is to automatically assess the impact of using new architectural features on the development projects (e.g., in terms of model complexity and required updates of the modeling tools) in order to assist system designers in planning their use in the models. The assessment is based on measuring the evolution of domain-specific meta-models, architectural models and system design requirements related to relevant architectural features.

Method: We performed a series of case studies focusing on the domain-specific meta-model, architectural models and system design requirements from the automotive domain. On the one hand, the case studies helped us to understand relevant industrial contexts for our research problems and develop our methods using constructive research methodology. On the other hand, the case studies helped us to empirically validate the results of our methods.

Results: We developed three new methods and software tools for automated impact assessment. The first method and the tool (*QTool*) show the complexity increase in the architectural models after adding a set of new features to the system. The second method (*MeFIA*) and the tool (*ARCA*) assess the impact of using these features in the system on the used modeling tools. Finally, the third method and the tool (*SREA*) identify a subset of design requirements that are affected by the use of the new features.

Conclusion: We showed in practice that our methods and tools enable faster use of new architectural features in the development projects. More concretely, we showed that quantitative analysis of evolution of domain-specific meta-models, architectural models and system design requirements related to new architectural features can be a valuable indicator of which features shall be used in the system and what is their impact on the development projects.

Acknowledgment

First and foremost, I would like to express my deepest gratitude to Prof. Miroslaw Staron, my main supervisor. I was very fortunate to work with Miroslaw for my master thesis. This served as an initial spark that lit my desire to pursue a PhD. Miroslaw's guidance and constant feedback before and during my PhD studies was invaluable for my development as a researcher.

Then, I would like to thank my co-supervisors Prof. Matthias Tichy and Prof. Jörgen Hansson, whose ideas and comments significantly improved the quality of my research and included publications. I would also like to thank my managers at Volvo Cars Stefan Andreasson and Hans Alminger for steering my development as an engineer and for supporting my wish to pursue a PhD.

There are a few other people from Volvo who I owe a great debt of gratitude. I am very grateful to Urban Kristiansson, now retired Senior Technical Advisor, for guiding me through the process of becoming a part of Volvo's Industrial PhD Program. I am also very grateful to my colleagues Nicklas Fritzson and Ola Reiner who kept this project alive with their ideas about the application of the research results. Finally, I am very grateful to Martin Nilsson whose industrial guidance while I was still a master student at Volvo shaped an important part of my PhD project.

Additionally, I would like to thank Corrado Motta, a former master student and now my colleague at Volvo Cars, and Maxime Jimenez, a former intern at Chalmers, for contributing to my project in two studies. Their work was very important for connecting the pieces of my thesis. I would also like to thank my colleagues from the AUTOSAR consortium who were always willing to discuss the results of my studies and offer suggestions for future work. In particular, I am grateful to Dr. Joakim Ohlsson from Volvo AB, Johan Ekberg from ArcCore, Anders Kallerdahl and Istvan Horvat from Mentor Graphics, Uwe Honekamp from Vector, and Dr. Tom Galla from Elektrobit.

At the end, my inexpressible appreciation goes to my family who are the most important thing in my life. First and foremost, I want to thank my wife Bojana for her encouragement in my moments of doubt and for sharing my moments of success. Without your sacrifices, finishing this thesis would not be possible. I am also very grateful to my parents Slobodan and Mirjana who were always there to support my personal and professional decisions in life.

The research presented in this thesis was conducted within the QuaSAR@car research project which is funded by Volvo Cars, as part of the Volvo Industrial PhD Program (VIPP), and by Swedish Governmental Agency for Innovation Systems (VINNOVA), under the grant no. 2013-02630.

List of Publications

Included publications

This doctorate thesis is based on the following publications:

[A] D. Durisic, M. Nilsson, M. Staron and J. Hansson, "Measuring the Impact of Changes to the Complexity and Coupling Properties of Automotive Software Systems", *Journal of Systems and Software (JSS), vol. 86, no. 5, pp. 275-1293, 2013*

[B] D. Durisic, M. Staron, M. Tichy, J. Hansson, "Addressing the Need for Strict Meta-Modeling in Practice - A Case Study of AUTOSAR", *Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD), pp. 317-322, 2016*

[C] D. Durisic, M. Staron, M. Tichy, J. Hansson, "Assessing the Impact of Meta-Model Evolution - A Measure and Its Automotive Application", *Journal of Software and Systems Modeling (SoSyM), pp. 1-27, 2017*

[D] M. Jimenez, D. Durisic, M. Staron, "Measuring the Evolution of Meta-Models - A Case Study of Modelica and UML Meta-Models", *Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development (MODELSWARD), 2017*

[E] D. Durisic, M. Staron, M. Tichy, "Identifying Optimal Sets of Standardized Architectural Features - A Method and its Automotive Application", *Proceedings of the 11th International Conference on Quality of Software Architectures (QoSA), pp. 103-112, 2015*

[F] D. Durisic, M. Staron, M. Tichy, J. Hansson, "ARCA - Automated Analysis of AUTOSAR Meta-Model Changes", *Proceedings of the 7th International Workshop on Modeling in Software Engineering (MiSE), pp. 30-35, 2015*

[G] C. Motta, D. Durisic, M. Staron, "Should We Adopt a New Version of a Standard? - A Method and its Evaluation on AUTOSAR", *Proceedings of the 17th International Conference on Product-Focused Software Process Improvement (PROFES), pp. 127-143, 2016*

[H] D. Durisic, C. Motta, M. Staron, M. Tichy, "Co-Evolution of Meta-Modeling Syntax and Semantics in Architectural Domain-Specific Modeling Environments - A Case Study of AUTOSAR", *Proceedings of the 20th International Conference on Model Driven Engineering Languages and Systems (MODELS), 2017*

Additionally, case study background of the thesis described in Chapter 2 is based on the the first five sections of the following book chapter:

- D. Durisic, "AUTOSAR", *Chapter in Automotive Software Architectures book, Springer, 2017*

Other publications

The following publications are published but not appended to this thesis due to their content overlap or smaller relevance to the research questions:

- D. Durisic, M. Staron and M. Nilsson, "Measuring the Size of Changes in Automotive Software Systems and their Impact on Product Quality", *Proceedings of the 12th International Conference on Product Focused Software Development and Process Improvement (PROFES), pp. 10-13, 2011*
- D. Durisic, M. Staron, M. Tichy, J. Hansson, "Evolution of Long-Term Industrial Meta-Models - An Automotive Case Study of AUTOSAR", *Proceedings of the 40th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), pp. 141-148, 2014*
- D. Durisic, M. Staron, M. Tichy, J. Hansson, "Quantifying Long-Term Evolution of Industrial Meta-Models - A Case Study", *Proceedings of the International Conference on Software Process and Product Measurement (MENSURA), pp. 104-113, 2014*

Contents

Chapter 1

Introduction

Model-Driven Engineering (MDE) [22] is a widely used methodology in the development of large software systems. The main benefits of MDE are the facts that it raises the level of abstraction [3] and employs tools to enable automation in the development process [2]. Raising the level of abstraction contributes to the reduced complexity of the system and increased understandability of the interplay between the system's components. Therefore, MDE is especially useful in the development of large systems that consist of many complex parts which have to communicate with each other [10]. Automation contributes to the increased development speed.

Automotive software system is a good example of a large software system where MDE facilitates the development process. One reason for this lies in the highly increased size and complexity of automotive systems in the past decade. Today, one premium car consists of more than 150 computers (referred to as ECUs - Electronic Control Units) responsible for executing more than 1 gigabyte of on board binary code, compared to only 50 ECUs and 10 megabytes of binary code ten years ago [18]. This trend of increasing complexity of the automotive software systems is expected to continue [26] driven by the new functionalities that are expected from modern cars, e.g., autonomous drive and car-to-car communication.

Another reason for using MDE in the automotive domain is the highly distributed development of software for different ECUs, which is usually developed by different suppliers. Therefore, using formalized models for exchanging information between different actors in the development process, and modeling tools for reading and editing the models in the automated way contributes to the increased development speed. An alternative approach would be manual work with the non-formalized artifacts of the system description which would, given the complexity and size of the system, be hardly feasible.

Other examples of large software systems can be found in the avionics domain, where the software for military jets today has up to 50 million lines of code, and the software for the newest passenger planes almost 100 million lines of code (comparable to some luxury cars) [18]. Similar to the automotive software systems, other large software systems are usually also composed of functionally diverse components which are developed by unrelated teams. This additionally increases the complexity of the development process.

In the area of software architectural design, the methodology of MDE is built around three main artifacts in the development process: domain-specific meta-models, architectural models and system design requirements. The architecture of the system is defined in the architectural models, which are developed in one or more architectural modeling tools. In order to be able to exchange the models between modeling tools used by different actors in the development process, the tools are based on a common domain-specific meta-model that defines syntax for the models, thus assuring the tooling interoperability. Finally, the semantics of the models is usually defined in the natural language specifications, which may consist of a number of design requirements explaining the use of one or more modeling elements.

Despite being used in industry for many years, employing MDE in practice brings a number of challenges related to the evolution of the MDE artifacts and their impact on each other. Some of these challenges are well covered in the existing literature, such as the coupled evolution of models and meta-models related to the automated model updates according to the new meta-model versions [28]. Certain challenges, however, received less attention, such as the impact of meta-model evolution on the compliant modeling tools and the complexity of the models. This is important for planning updates of the tools in order to support new modeling features in the meta-model, and prioritizing testing areas in the system upon using these features in the models. Therefore, the existence of such methods for automated impact assessment has a potential to increase the speed innovation in the development projects.

The main goal of this thesis is to develop methods and tools for managing architectural updates, that require updates of the modeling language, in the development of large software systems. This management includes the assessment of impact of using a new architectural feature in the models (e.g., new communication protocol) on the modeling tools used in the development process, in terms of updating effort, and existing models, in terms of increased complexity. The tooling impact assessment is particularly important if software is developed in a distributed environment, like in the automotive domain, where architectural models are exchanged between multiple actors in the development process. This is because each actor may use different modeling tools, and all of them should be considered in the assessment.

In order to achieve our goal, we performed a series of case studies of meta-models, architectural models and design requirements. For example, we examined the architectural models from Volvo Cars and the domain-specific meta-model and the standardized requirements from the AUTOSAR standard [6], a reference architecture and methodology used in the automotive domain. We used these case studies as part of the constructive research methodology to develop, evaluate and validate our methods for analyzing the evolution of these three MDE artifacts. For example, we performed one case study to understand the organization of domain-specific meta-models in order to define a method for measuring meta-model evolution, and another case-study to evaluate and validate the method using one or more industrial meta-models.

The main results of this thesis are three methods for automated impact assessment of using new architectural features in large software systems on the development projects, each based on one or two software measures. This is based on our hypothesis that simple measures of change in the domain-specific

meta-models, architectural models and design requirements can produce accurate early indicators of which architectural features shall be used in the system, and estimate their impact on the development projects.

The first method, realized in the *QTool*, shows the complexity increase in the architectural models after adding a set of new features to the system. The method is based on the two measures of model complexity and coupling. The second method, named *MeFIA* and realized in the *ARCA* tool, assesses the impact of supporting different features on the used modeling tools. The method is based on the measure of meta-model change (*NoC*) and it can also show concrete meta-model changes caused by a particular feature and relevant to a particular actor in the development process. Finally, the third method, realized in the *SREA* tool, identifies the subset of design requirements, that provide semantics to the modeling elements, affected by the analyzed feature. The method can also identify the most unstable requirement specifications based on the measures of requirements change and maturity index (*RMI*).

We can see that each of the three methods analyzes the evolution of one of the three MDE artifacts, i.e., domain-specific meta-models, architectural models and system design requirements. The methods can also perform impact assessment of this evolution on the development projects, i.e., the first method in terms of complexity increase in the models upon using the new architectural features, the second method in terms of updating efforts of the used modeling tools to support the new features, and the third method in terms of time required to understand how to use the new features in the models. Therefore, the methods complement each other and should be used together by the system designers in order to decide which new features shall be used in projects, and accurately estimate their impact on the development projects.

For example, the *ARCA* tool can be used first to assess the feasibility of supporting one feature in the modeling tools in the project time-frame. If it is considered feasible, the *ARCA* and *SREA* tools can then be used for planning the required updates in the tools and understanding the relevant requirements specifications, respectively, related to this feature. Finally, the *QTool* can be used to indicate which parts of the system are mostly affected by the use of this features for prioritizing testing areas or revising the system's architecture.

In order to validate our methods in practice, we applied them in the automotive domain on the AUTOSAR artifacts. The choice of the automotive domain is justified as automotive software systems represent large systems which are developed, including the architectural design, fully according to MDE. The choice of AUTOSAR is justified due to its long industrial use (more than 10 years) by the majority of car manufacturers, and its large size. For example, the newest release of the AUTOSAR meta-model exceeds 10000 meta-classes and 35000 meta-model changes in comparison to the previous release, and more than 20000 requirements in the AUTOSAR specifications our of which at least 2000 are changed in every release.

We first found that, using the *QTool*, it is possible to verify that certain quality attributes of the AUTOSAR models have not deteriorated and to identify new testing areas in the system. We then found that the *ARCA* tool can provide a preliminary indicator of effort needed to update AUTOSAR-based modeling tools in order to support new AUTOSAR features. Finally, we found that, using the *SREA* tool, we can support system designers to faster under-

stand how to use the new features in the architectural models. We used these findings to validate our hypotheses described above.

This rest of Chapter 1 is structured as follows: Sections 1.1 and 1.2 provide theoretical background related to modeling/meta-modeling and software measurement, respectively, and how it is used in this thesis. These two areas of software engineering are particularly important as our methods are based on software measures applied on different modeling artifacts. Section 1.3 defines our research questions and describes the contribution of our studies. Section 1.4 describes the methodology we employed to obtain the results. Finally, Section 1.5 summarizes our conclusions and describes possible future work.

Chapter 2 describes automotive modeling environment and the role of the AUTOSAR standard, as our main unit of analysis, in it. Chapters 3-10 present the individual papers included in this thesis. Each paper is independent and represents one study that addressed one or several research questions.

1.1 Modeling and meta-modeling

In this section, we first explain the theory behind modeling and meta-modeling following the MDE approach in Section 1.1.1. We then describe how meta-modeling is used in different domains, such as automotive, in Section 1.1.2. After that, we show the use of meta-modeling in the architectural design of one system and clarify the term "architectural feature" in Section 1.1.3. These three sections define the terms related to modeling and meta-modeling (indicated in bold) which are used in the rest of the thesis. Finally, we explain the use of modeling and meta-modeling in this thesis in Section 1.1.4.

1.1.1 Theory of modeling and meta-modeling

Modeling plays an important role in the development of large software systems because it reduces their complexity by raising the level of abstraction. This abstraction is achieved by specifying what the system does rather than how it does this [3]. Models and meta-models are the two most important concepts involved in any modeling environment. Based on the definitions of Bézivin et al. [9], a **model** represents a simplified representation of a software system that has been created for a specific purpose, whilst a **meta-model** represents the model of the language this system model is expressed in.

The prefix "meta" is of Greek origin meaning "after" or "beyond", and in general it indicates that certain concept lies above the original concept; for example, the meta-model represents "a model of the model" and the meta-meta-model represents "a model of the meta-model". Applying this logic several times creates a **meta-modeling hierarchy**, which is also referred to as the meta-pyramid [19]. The meta-modeling hierarchy of MOF [30] standardized by the Object Management Group (OMG) [33] is considered a *de facto* standard in meta-modeling and it consists of the following layers:

1. The **M3 layer**: MOF meta-meta-model that defines modeling concepts
2. The **M2 layer**: meta-model that defines language specifications
3. The **M1 layer**: model that defines application meta-data

4. The **M0 layer**: objects that define application data

The Meta-Object Facility (MOF) [30] resides at the top of the hierarchy. This is a meta-meta-model that defines the general modeling concepts used by meta-models on the *M2* layer. A frequently used meta-model on the *M2* layer is UML (Unified Modeling Language). The actual UML models reside on the *M1* layer and their actual execution at run-time resides on the *M0* layer. An example of the meta-modeling hierarchy is depicted in Figure 1.1.

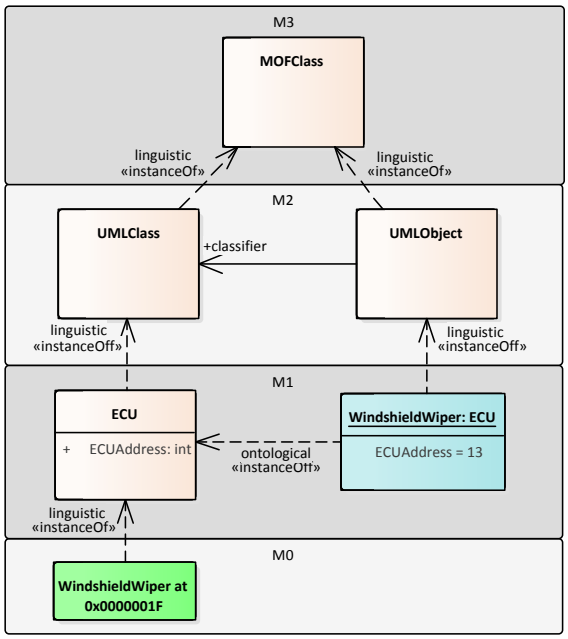


Figure 1.1: Example of the meta-modeling hierarchy

The layers of MOF are connected by the **instantiation** mechanism, i.e., elements of each layer represent instances of the elements of the layer above, except for the top layer *M3* whose elements are considered to be instances of themselves. For example, *ECU* on the *M1* layer is an instance of *UMLClass* on the *M2* layer. This type of instantiation is referred to as **linguistic instantiation**, and it is used to provide the instantiating elements with general language types (e.g., classes, objects, attributes). In addition to the linguistic instantiation, **ontological instantiation** is used to provide the instantiating elements with semantic classifiers (e.g., *WindshieldWiper* is an *ECU*).

In both case, the instantiating element defines the characteristics of the instance element, and the instance element defines the specific details of these characteristics. The *instanceOf* relationship, however, is not transitive, e.g., the *M1* model element will only receive characteristics from the *M2* meta-model elements and not from the *M3* meta-meta-model elements [5].

According to the **strict meta-modeling** principle [4], all model elements on one layer of a meta-modeling hierarchy are instances of the meta-model elements of the layer above, except for the top layer. No relationships other than *instanceOf* are allowed to cross the layer boundaries, and no *instanceOf*

relationships are allowed within one layer. If these conditions are not met, we refer to the meta-modeling as loose. An example of **loose meta-modeling** can be seen in Figure 1.1 where the *instanceOf* relationship between the *WindshieldWiper* and *ECU* resides entirely on the *M1* layer.

1.1.2 Domain-specific modeling and meta-modeling

Meta-modelling plays an important role in the development of description languages suitable for modeling systems in specific domains [37], e.g., automotive, telecommunications and avionics. A **domain-specific model** represents an abstract representation of the system of a particular domain, while a **domain-specific meta-model** defines the syntax and the semantics of the domain-specific models instantiating this meta-model [31].

Four important concepts of domain-specific meta-models can be distinguished: abstract syntax, concrete syntax, static semantics (well-formedness) and dynamic semantics (or just semantics). The **abstract syntax** describes the structural essence of the meta-model, e.g., elements and their relations independent of the representation of the actual models. An example can be seen in Figure 1.1 in the definition of the *ECU* element and its potential relation to other elements, e.g., *Signal*s and *Bus*es. The **concrete syntax** specifies the representation of the model instances, e.g., *WindshieldWiper*, in the models. These representations include graphical notations and XML, as a commonly used format for exchanging models between different modeling tools. There can be more than one concrete syntax for one abstract syntax, and vice versa.

The **static semantics** impose a set of constraints on the abstract syntax. For example, each *ECU* needs to have an *ECUAddress*. This can be achieved by specifying different constraints, e.g., in the form of multiplicities of the attributes and relationships or using OCL (Object Constraint Language) [32]. Finally, the **dynamic semantics** provides meaning to the syntax notation of the meta-model and it can be formal or informal (e.g., natural language specifications). For example, "An ECU represents one micro-controller in the system." informally describes the semantics of the *ECU* element. One domain-specific meta-modeling environment usually specifies all four concepts [24], i.e., the meta-model itself specifies the abstract syntax and the static semantics, while the concrete syntax (e.g., XML) for the models and their dynamic semantics may be specified in the supporting natural language specifications.

Domain-specific meta-models are often linguistically instantiated from a general-purpose meta-model such as MOF or UML. In case of UML-based domain-specific meta-models, UML **profiles** with the defined **stereotypes** and **tag definitions** can also be used for customizing the abstract syntax and the static semantics of the UML meta-model for a specific domain.

One domain-specific modeling environment may contain an arbitrary number of layers; in practice, there can be more than the four layers defined by MOF. These layers need to co-evolve in order to support the addition of new modeling and meta-modeling features [14]. For example, in order to express new modeling features on the *M1* layer, the *M2* layer need to evolve in order to describe how to model these new features. Similarly, evolution of the *M2* layer may require evolution of existing models of the *M1* layer in order to maintain conformity between the *M1* and *M2* layers.

System modelers of one domain-specific modeling environment usually rely on **software modeling tools** (CASE - Computer Aided Software Engineering tools) to create and update models and generate code based on them. Since the development of large software systems often involves multiple **actors** (**design roles**) potentially using different modeling tools in the development process, a smooth exchange of models between these roles can be somewhat challenging. Nevertheless, a smooth exchange can be enabled by defining and gaining consensus from all roles for a domain-specific meta-model, which is then fundamental to the development of all tools used in a specific modeling environment. This is based on the assumption that if two modeling tools adopt the same model structure defined by the meta-model, they can exchange software models that comply with this meta-model [2].

Since the modeling tools are based on a commonly-accepted domain-specific meta-model, its evolution may significantly impact all the tools in a specific modeling environment. Such tools tend to be developed based on their own meta-models (e.g., to support graphical representations) so the evolution of the domain-specific meta-models directly impacts the importers and exporters of the compliant models, as well as the meta-models used by these tools.

1.1.3 Architectural modeling and architectural features

Software architecture represents a set of design decisions made about a system (not all design decisions are architectural, though). A model that captures some or all of these design decisions can be regarded as the **architectural model** [41]. An architectural model defines a number of architectural components responsible for the execution of different system functionalities.

Based on these definitions, the possibility of utilizing the architectural components and their interactions in a specific way to achieve certain semantics can be considered as an **architectural feature**. For example, the possibility of modeling communication between two ECUs of specific ECU addresses. The architectural models and their features are expressed using a general modeling language, which has both the linguistic and ontological instantiations. For example, in order to model the concrete ECUs and their addresses, a domain-specific meta-model must define the "*ECU*" class with "*ECUAddtess*" attribute, as shown in Figure 1.1.

1.1.4 Modeling and meta-modeling in this thesis

As already explained, in this thesis we focus on analyzing the evolution of three main artifacts used in domain-specific meta-modeling environments: meta-models, models, and design requirements. Meta-models define abstract syntax and static semantics for the models which ontologically instantiate them using a concrete syntax, while design requirements provide informal dynamic semantics to the models. Table 1.1 show the focus of our studies and studied artifacts in different papers:

As we can see, the majority of our studies focus on the analysis of the AUTOSAR meta-modeling environment, except the one described in Paper D that analyzes UML and Modelica meta-models. AUTOSAR meta-model is used as a basis for the architectural design of automotive software systems and

Table 1.1: Study focus and studied artifacts

Focus	Artifact	Paper
Analyzing domain-specific meta-modeling environments with respect to MOF	AUTOSAR meta-model	B
Analyzing the evolution of models	AUTOSAR models	A
Analyzing the evolution of meta-models	AUTOSAR meta-model	C, E, F, H
Analyzing the evolution of meta-models	UML and Modelica meta-models	D
Analyzing the evolution of system design requirements	AUTOSAR design requirements	G, H

the development of AUTOSAR modeling tools (CASE tools), i.e., it defines architectural models and their features. It is defined as a linguistic instance of UML. AUTOSAR models represent ontological instances of the AUTOSAR meta-model and linguistic instances of UML. As explained in Paper B, both the AUTOSAR meta-model and the AUTOSAR models reside on the *M2* layer of the MOF hierarchy, despite the fact that AUTOSAR defines five meta-modeling layers (see Chapter 2).

Due to the fact that AUTOSAR artifacts represent the most important units of analysis in this theses, we dedicated Chapter 2 to the detailed definition of the AUTOSAR standard and its role in the development of automotive software architectures. In this chapter, one can also find detailed description of the AUTOSAR meta-model and examples of the AUTOSAR models in XML and textual AUTOSAR design requirements.

Together with the analysis of the AUTOSAR meta-model, as a domain-specific meta-model used for defining architectural models, we analyzed two additional meta-models of a different kind. The first one is the meta-model of Modelica [34], which is used for defining the behavioral models of complex systems containing, e.g., mechanical, electrical and electronic components. The second one is the meta-model of UML [42], which represents a general-purpose meta-model used for modeling a number of aspects, such as architecture (e.g., using UML class diagrams) and behavior (e.g., using UML sequence or state machine diagrams), of a wide variety of software systems. It represents a direct linguistic instance of the MOF meta-model, and it can also be used as a meta-meta-model for defining further domain-specific meta-models.

1.2 Software measurement

In this section, we first explain the theory behind software measurement in Section 1.2.1. We then describe how to perform the measurement process in Section 1.2.2. These two sections define the terms related to software measurement (indicated in bold) which are used in the rest of the thesis. Finally, we explain the use of software measurement in this thesis in Section 1.2.3.

1.2.1 Measurement theory

Measurement in software engineering plays an essential role not only to assessing a variety of quality attributes of the software system, such as reliability, maintainability and efficiency [40], but also to estimating the implementation cost and effort to support different features in the system. According to the measurement theory [15], **measurement** is a process in which numbers (or symbols) from the mathematical world are assigned to different entities from the empirical world to describe the entities according to defined rules. A **measure** (also referred to as **metric** [16]) represents a variable to which a value is assigned as a result of the measurement [38].

The measurement theory describes how to construct measures and formalize their mapping from the empirical to the mathematical world, i.e., empirical relations between entities are mapped to mathematical relations so that they can be analyzed. For example, if two software systems (A and B) are related by the empirical relation "more complex than", we can define a measure of their complexity (c), where measurement results are related by the mathematical relation ">". The mapping between the empirical and mathematical relations, therefore, is defined as:

$$c(A) > c(B) => A \textit{ more complex than } B \tag{1.1}$$

Measurement results can be represented on different scales and different relations between the results are possible depending on the scale [15]:

- *Nominal scale* - only a relation of equivalence possible (A = B)
- *Ordinal scale* - a nominal relation + greater/smaller than (A > B)
- *Interval scale* - an ordinal relation + difference computation (A − B)
- *Ratio scale* - an interval relation + ratio computation (A / B)
- *Absolute scale* - like ratio, but for measurements with a predefined minimum value (e.g., zero) which are mostly used for counting entities

In order to structure software measures according to their objectives, **Goal Question Metric** (GQM) approach proposed by Basili et al. [8] can be used. GQM defines the measurement as a mechanism that helps to answer a variety of questions about the software process and products. It defines the measurement model on three levels: Conceptual (goals), Operational (questions) and Quantitative (metrics), as shown in Figure 1.2:

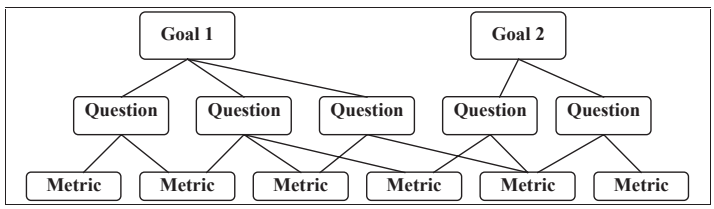


Figure 1.2: The Goal Question Metric (GQM) levels [8]

The **goal** is defined for one object (e.g., a product or process) with respect to its quality attributes for a specific purpose (e.g., an evaluation) and from a specific perspective (e.g., that of a system designer). A set of **questions** for one goal is used to specify how the goal will be assessed by characterizing the object to be measured. Finally, the **metrics** represent the quantitative data associated with every question that enable the question to be answered.

1.2.2 Measurement process

The *ISO/IEC 15939* standard for the **measurement process** in software engineering [39] defines the process as a set of activities that are required to specify: (i) what **information need** is required for the measurement, (ii) how the measures are made and measurement results are analyzed, and (iii) how the results are validated. Additionally, the measurement process specifies how to build the measurement products, although this area is beyond the scope of this thesis. A simplified measurement process is shown in Figure 1.3:

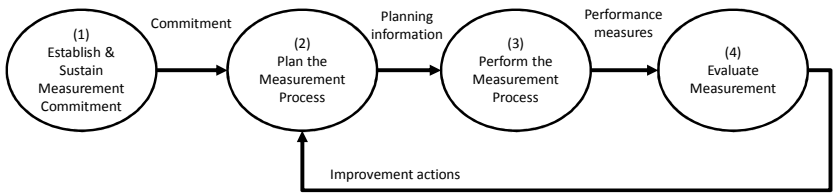


Figure 1.3: Measurement process activities [39]

Activity (1) defines the **scope of the measurement** and who will execute it. Activity (2) elaborates on the **measurement plan**, such as what is to be measured (i.e., which entities and their quality attributes), what information is needed (i.e., the reason for the measurement), which measures will be used and on what scale, and the criteria for evaluating the measurement results. Activity (3) describes the **data collection** and **data analysys**. Finally, activity (4) describes the **evaluation** of the measures and the measurement process based on the defined criteria of activity (3). This process is defined as iterative in order to improve both the measures and measurement process based on the results of the evaluation.

One important segment of the planning activity (3) is to ensure the **definition of the measures** in order to avoid different interpretations of how the measurement has been done (see the measurement errors of different implementations of the commonly known lines of code measure [36]). The measures will be defined based on the **conceptual model** (also referred to as the **data model**), which is used to describe the entities in the empirical world [20] to ensure that the metrics can satisfy the required information need.

Software measures are usually defined using either set theory or algebra expressions. In order to prevent a definition of a measure using one of these two approaches from becoming too complex, however, alternative approaches can be taken, e.g., using pseudo-code snippets. For example, the complexity (c) of one software component (x) that can be used as an indicator of the number of faults in it's code, can be defined using algebra as follows:

$$c(x) = \sum_{i=1}^{n} r_i(x) * \sum_{i=1}^{n} t_i(x);$$

$$r_i(x) = \begin{cases} 1, & \text{if x receives } sig_i \\ 0, & \text{otherwise} \end{cases} ; t_i(x) = \begin{cases} 1, & \text{if x transmits } sig_i \\ 0, & \text{otherwise} \end{cases}$$

where n represents the total number of signals and sig_i the signal with serial-number i. Using set theory, this same result can be achieved by defining two sets $Sin(x)$ and $Sout(x)$:

- $Sin(x) = \{sin_1(x), sin_2(x), ..., sin_\alpha(x)\}$ - a set of signals received by software component x.
- $Sout(x) = \{sout_1(x), sout_2(x), ..., sout_\beta(x)\}$ - a set signals transmitted by software component x.

The complexity would then be calculated as follows:

$$C(x) = |Sin(x)| * |Sout(x)|$$

Finally for the purpose of completeness, the simplified corresponding pseudo-code could look like this:

```
int Complexity(Component x)
{
   int Sin = 0;
   int Sout = 0;

   foreach (Signal in ReceivedSignals(x))
      Sin = Sin + 1;

   foreach (Signal in TransmittedSignals(x))
      Sout = Sout + 1;

   return Sin * Sout;
}
```

An important part of the measurement process is to validate the defined software measures. Two different types of validation can be performed [11]: a theoretical validation to answer "*Are we measuring the right attribute?*" and an empirical validation to answer "*Is the measure useful?*".

The **theoretical validation** ensures that a measure does not violate the properties of the measured entity [23]. This can be achieved by assessing whether the measure satisfies certain theoretical criteria. For example, there must be at least two entities for which the measure yields a different result, measuring the same entity twice yields the same results or measuring two entities can yield the same result. Additionally, Briand et al. [12] classify the measures according to five attributes (size, length, complexity, coupling and cohesion) and define a set of properties required to measure each attribute. These properties can be used to group measures according to different properties and validate that they indeed measure an intended attribute.

The **empirical validation** ensures that the measurement results reflect the entities of the real world [23]. This can be achieved by discussing the results with experts who work with the measured entity in order to ensure

that they are consistent with their expectations (e.g., code complexity should decrease after re-factoring). Statistical analysis can also be used to validate the relationship between two attributes of the measured entity using their historical values. This is useful in situations where the value of one attribute can be predicted by measuring the value of another attribute. An example of this is the use of a correlation analysis based on historical data to empirically validate the link between code complexity and the number of faults.

1.2.3 Software measurement in this thesis

The results presented in this thesis rely heavily on the use of software measures and measurement results as all papers, except from Paper B, define and/or use one or more software measures. The meta-models, architectural models and system design requirements represent the scope of the measurements, and our main information need was to assess their evolution with respect to a number of properties, including changes, size, complexity and coupling. Since available, off-the-shelf measures do not use the specific modeling characteristics of the automotive domain, we defined our own set of measures for this purpose. A summary of the most important measures used in our studies is shown in Table 1.2.

Table 1.2: The most important measures used in the studies of this thesis

Measure name	Measure goal	Defined	Used
Complexity measure	Monitoring the complexity evolution of the automotive software systems.	Paper A	Paper A
Package coupling measure	Monitoring the coupling evolution of the automotive software systems.	Paper A	Paper A
Number of (meta-model) changes measure (*NoC*)	Estimating the effort/cost of adopting a new meta-model version or a subset of the new features it supports.	Paper C	Papers C, D, E, F, H
Number of changed requirements	Monitoring the evolution of system requirements.	Paper G	Papers G, H
Requirements maturity index (*RMI*)	Monitoring the stability of requirements changes in relation to the past releases.	Paper G	Paper G

Some studies also include additional more detailed measures. For example, in addition to the results of the *NoC* measure, Papers D and F show the results of the *Number of added/modified/removed elements* (elements can be classes, attributes and connectors) and the *Number of elements* measures. Another example can be found in Paper H where, in addition to the results of the *Number of changed requirements* measure, the results of the *Number of added/modified/removed requirements* measure are used.

The majority pf our measures are developed following the GQM approach based on the appropriate data models. Clearly defining both the goals and

questions for each study enabled us to re-use the logic behind certain measures used for measuring one entity, in order to address similar questions and goals about another entity. For example, the goal of our *Number of changed elements* measure in Paper F was to monitor the evolution of domain-specific meta-models. Similarly, to the goal our *Number of changed requirements* measure in Paper G was to monitor the evolution of requirement specifications.

Some of our measurement results are presented on the ratio scale (e.g., model *Complexity* and *Package coupling* measures) whilst others are shown on the absolute scale (e.g., *NoC* and the *Number of changed requirements*). The majority of measures are defined using either algebra or pseudo-code snippets. For example, we defined the *Complexity* and *Coupling* measure of the architectural models used in Paper A using algebra and the *NoC* measure for domain-specific meta-models used in Paper C using pseudo-code snippets. However, some measure are defined in words, e.g., the *Number of changed requirements* and *RMI* measures used in Paper G.

The process of data collection was fully automated as we developed software tools to measure the properties of architectural meta-models, models and requirements based on the appropriate data model. All the presented measures are evaluated by calculating them on industrial project data, e.g., from Volvo Cars and the AUTOSAR consortium. The *Complexity* and *Coupling* measures for monitoring the evolution of architectural models were calculated on a number of software components and ECUs from the two evolving models at Volvo Cars. The *NoC* measure was calculated on a number of AUTOSAR meta-model releases and the new features they support, and a set of releases of Modelica and UML meta-model. Finally, the *Number of changed requirements* and *RMI* measure were calculated on a number of AUTOSAR requirement specification from a set of chosen AUTOSAR releases.

The empirical validation of the measures, except for the *RMI measure*, was done using one of the following two approaches: The first approach was to match the measurement results with the expectations from the experts in the field and/or available documentation. The second approach was to use statistical methods, such as correlation analysis, in order to analyze the relationship between the measurement results and the attributes of the measured entity.

The *Complexity* and *Coupling* measures for monitoring the evolution of the architectural models were also validated theoretically based on the properties of complexity and coupling measures defined by Briand et al. [12].

1.3 Research questions and contributions

The goal of this thesis was to develop methods and tools for managing architectural updates in the MDE development of large software systems. These architectural updates are usually manifested in a form of new architectural features used in the development projects. In order to achieve our goal, we focused our studies on the automotive domain, considering automotive software system as a good example of a large software system developed following the MDE approach. Therefore, we defined the following main research question:

RQ: How to automatically assess the impact of using new architectural features in the system on automotive software development projects?

Using new architectural features in the system, that require updates of the modeling language, causes the evolution of domain-specific meta-models, architectural models and system design requirements. The evolution of these three MDE artifacts, however, has significant impact on the development projects. The evolution of meta-models requires updates of the used modeling tools and possibly existing models. The evolution of architectural models usually requires verification and validation of the entire system. The evolution of design requirements requires detailed inspection of the requirement specifications for the correct use of new features in the models. Therefore in order to be able to assess the impact of using new architectural features on the development projects, we needed to analyze the evolution of architectural meta-models, models and design requirements, each representing one direction in our study.

In order to address our main research question, we divided it into fourteen smaller research questions, each addressed in one of our eight studies/papers. These smaller research questions, including a short description of our contributions in each paper, are presented in Table 1.3.

Table 1.3: Research questions and contributions

No.	Research question	Contribution/finding	Paper
RQ1	How can the complexity increase of architectural models and their components be monitored during the evolution of large software systems?	Two measures are needed to monitor the complexity evolution of automotive architectural models when new architectural features are added to the system - the measures of architectural complexity and coupling. *QTool* can be used for combined analysis of results of these two measures.	Paper A
RQ2	What are the consequences of UML based loose metamodeling in the automotive domain?	The main consequence is that not all semantics can be conveyed between metamodeling layers by means of modeling, e.g., which defined stereotypes are applicable to classes and which to associations. In practice, this is solved by the modeling tools by providing means to specify additional semantics.	Paper B
RQ3	What are the drawbacks of approaches for assuring strictness of the AUTOSAR meta-model?	The main problem with approaches assuring strictness is the lack of tool support and their relatively short and narrow use in industry.	Paper B

No.	Research question	Contribution/Finding	Paper
RQ4	What are the practical meta-modeling concerns of the automotive modeling practitioners?	One of the major practical concern of the automotive modeling practitioners is the impact of evolution of domain-specific meta-models on other artifacts in the development process.	Paper B
RQ5	How can the evolution of domain-specific meta-models be measured in order to accurately reflect the impact of meta-model changes on the modeling tools used by different actors in the development process?	A simple measure of meta-model change (*NoC*) can be used as a preliminary indicator of impact of new domain-specific meta-model versions on the used modeling tools.	Paper C
RQ6	What types of changes can be distinguished between different versions of the AUTOSAR meta-model?	Data model that captures all relevant meta-model changes for analyzing the evolution of the AUTOSAR meta-model.	Paper C
RQ7	How can the evolution of the AUTOSAR meta-model be quantified?	The *NoC* measure based on our data model for quantifying the evolution of the AUTOSAR meta-model.	Paper C
RQ8	How accurately can quantitative analysis of the AUTOSAR meta-model changes be used for predicting its impact on the AUTOSAR tools?	Statistically significant positive Spearman's correlation of 0.69 between the results of *NoC* and the actual effort needed to update the AUTOSAR based modeling tools.	Paper C
RQ9	What is the level of applicability of the measures of domain specific meta-model evolution and the underlying data-model defined in (Durisic et al., 2014) for monitoring the evolution of Modelica/UML meta-models?	The *NoC* measure and the underlying data-model are applicable for measuring the evolution of two additional meta-models of Modelica and UML.	Paper D
RQ10	How to assess the impact of different architectural features on the used domain-specific meta-models?	The *MeFIA* method can be used for assessing the impact of new architectural features on domain-specific meta-models.	Paper E

No.	Research question	Contribution/Finding	Paper
RQ11	How to identify the optimum set of features to be adopted based on the assessed impact?	The *MeFIA* method can be used for identifying optimal sets of new architectural features to be used in the development projects.	Paper E
RQ12	How to support modeling practitioners in analyzing changes between different versions of domain-specific meta-model related to different architectural features?	The *ARCA* tool realizing the *MeFIA* method can be used to support automotive modeling practitioners in deciding which new AUTOSAR features to use in the development projects.	Paper F
RQ13	How can we assure efficient adoption of new releases of standards in the development of large software systems by analyzing the evolution of standardized requirements?	The *SREA* tool can be used to support organizations in understanding which parts of the system will be mostly affected by the changes in the system design requirements.	Paper G
RQ14	How strong is the relation between the evolution of meta-modeling syntax and meta-modeling semantics?	Statistically significant positive Spearman's correlation of 0.63 between the results of the *NoC* measure (for meta-modeling syntax) and the *Number of changed requirements* (for meta-modeling semantics).	Paper H

The relation between our studies and research questions and how answers to our smaller research questions contribute to our main research question and general goal of this thesis are presented in Figure 1.4:

We divided our research area into three lanes, each lane dedicated to the analysis of evolution of one of the three main MDE artifacts - models, meta-models and design requirements (indicated as yellow, blue and green lanes in Figure 1.4, respectively). We first examined the evolution of architectural models in the yellow lane by investigating how to monitor the complexity increase of the automotive architectural models after certain architectural changes have been made. As the answer to RQ1, we found that two measures are needed for this purpose - the complexity and the coupling measure defined in Paper A. We also showed in the same paper the practical use of these two measures on a case of two evolving automotive architectural models. We did this by developing a method and the *QTool* implementing this method which can calculate the measures and perform combined analysis of the measurement results.

We then examined the evolution of meta-models in the blue lane. In Paper B, we analyzed the relevance of theoretical meta-modeling concepts in practice, i.e., strict vs. loose meta-modeling, in order to position our studies with respect to meta-modeling theory. We identified three consequences of loose meta-modeling related to the organization of meta-modeling layers, as the answer

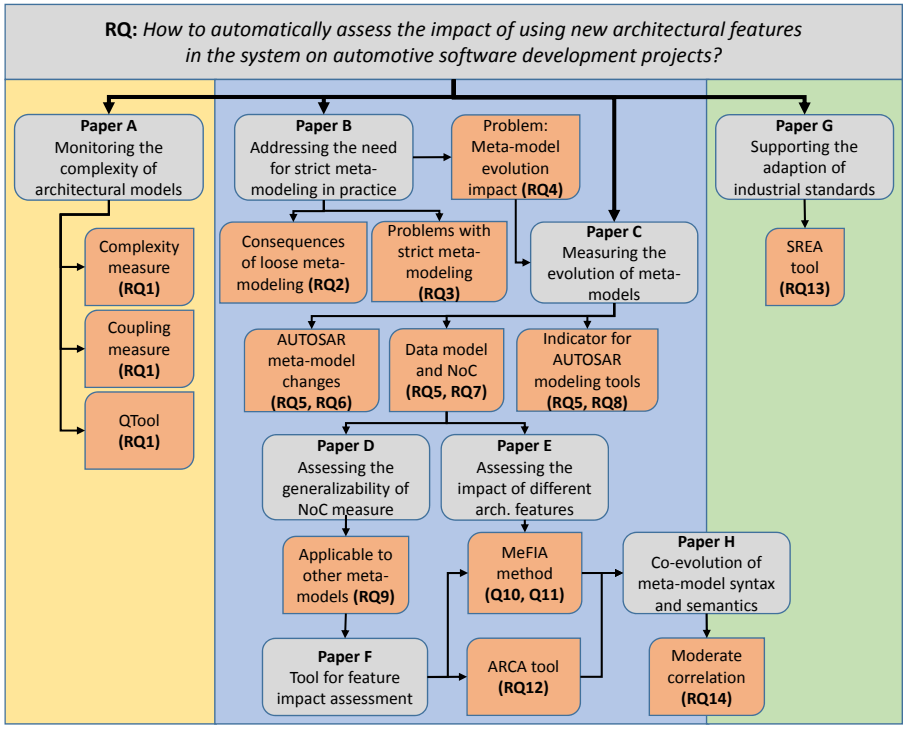


Figure 1.4: StudyDesign

to RQ2. We also identified the main problem of lacking tool support for the use of strict meta-modeling in practice, as the answer to RQ3. Finally as the answer to RQ4, we found that the evolution of domain-specific meta-models and their impact on the development projects is one of the major problems of the (meta-) modeling practitioners. This information served as a valuable input for our study described in Paper C which defined a measure of meta-model evolution (*NoC*) that can be used for preliminary impact assessment of new meta-model versions on the used modeling tools.

As the answer to RQ5 and more concrete RQ6, RQ7, and RQ8 defined in Paper C, we first identified meta-model changes that should be considered in the analysis of the AUTOSAR meta-model evolution (RQ5, RQ6). Considering these changes, we then defined a data model for the measurement and the *NoC* (Number of Changes) measure based on this data-model in order to quantify the evolution of architectural domain-specific meta-models (RQ5, RQ7). Finally, we validated the *NoC* measure by finding moderate to strong positive Spearman's correlation of 0.69 between its results and the actual effort needed to update AUTOSAR modeling tools according to the meta-model changes (RQ5, RQ8).

On the one hand, we used the *NoC* measure (and the underlying data model) in Paper D in order to assess its applicability to a wider range of meta-models. As the answer to RQ9, we found that our data model is able to capture relevant changes for monitoring the evolution of UML and Modelica meta-models. We also managed to calculate the *NoC* measure between different

releases of UML and Modelica. On the other hand, we used the *NoC* measure in Paper E to construct the *MeFIA* method that is able to assess the impact of a particular architectural feature on the used domain-specific meta-model. This is done by calculating meta-model changes related to this feature only, which represents the answer to RQ10. The *MeFIA* method is also able to identify optimal set of features to be adopted in the development projects based on their meta-model impact, which represents the answer to RQ11.

In order to enable industrial use of the *MeFIA* method in the automotive domain, we implemented the *ARCA* tool described in Paper F. The *ARCA* tool is able to automatically perform steps of the *MeFIA* method for a set of AUTOSAR features. Therefore, it provides the answer to RQ12 as it supports automotive engineers in analyzing the impact of different AUTOSAR features.

Finally, we examined the evolution of system design requirements in the green lane. We focused on the standardized requirements that provide semantics for the meta-modeling elements, as presented in Paper G. As the outcome of this study and the answer to RQ13, we constructed a method and the *SREA* tool implementing this method that can identify a subset of design requirements that are affected by the introduction of new architectural features.

Having the *SREA* and *ARCA* tools in place, we were also able to perform an additional study described in Paper H with the goal to investigate the relationship between the evolutions of meta-modeling syntax and meta-modeling semantics. As the outcome of this study and the answer to RQ14, we found a moderate positive Spearman's correlation of 0.63 between the evolutions of these two artifacts. This confirms the importance of analyzing the evolution of system design requirements together with the evolution of meta-models, and requires more effort from the meta-modeling practitioners in describing the syntactical changes in the meta-models.

Summarizing the results from all three lanes results in three new methods and tools (*QTool*, *ARCA* and *SREA*), one from each lane, that can be used for monitoring the evolution of architectural meta-models, models and system design requirements. The combined use of these methods and tools for assessing the impact of new architectural features on the development projects represents the answer to our main research question.

1.3.1 Industrial contribution

All results presented in this thesis are directly implemented at Volvo Cars by means of incorporating the *QTool*, *ARCA* and *SREA* tools into the company's change management process.

The *QTool* implements the complexity and coupling measures presented in Paper A and it is primarily used by the system architectural testers at Volvo Cars. The tool is used during the evolution of automotive architectural models in order to analyze the impact of added functionality on the complexity and coupling properties of different architectural components (e.g., sub-systems, ECUs and domains). If the results are unsatisfactory, some components need to be re-designed to reduce coupling and increase cohesion, e.g., by re-allocating functionality onto different ECUs. The results of the *QTool* are also used to indicate which parts of the system require more testing.

The *ARCA* tool implements both the *MeFIA* method and the measures

defined in Papers C, E, F and H (Paper F describes the *ARCA* tool itself) for monitoring of the evolution of the AUTOSAR meta-model. The tool is primarily used in one of the following three scenarios:

1. **To decide which AUTOSAR meta-model release shall be used**

   The analysis is primarily done by the AUTOSAR team at Volvo Cars with support from other teams (e.g., system architects and designers). The analysis includes impact assessment of adapting a new AUTOSAR release on the modeling tools used by different actors (design roles) in the development process. The aim of this analysis is to indicate if a new AUTOSAR release is feasible to be adopted in the development process.

2. **To decide which new AUTOSAR features shall be used**

   The analysis is primarily done by the AUTOSAR team at Volvo Cars with support from other teams (e.g., system architects and designers). The analysis includes impact assessment of adapting each new AUTOSAR feature on the modeling tools used by different design roles in the development process. The aim of this analysis is to indicate if a new AUTOSAR feature is feasible to be adopted in the development process.

3. **To influence standardization of the AUTOSAR meta-model**

   This analysis is done by the AUTOSAR team at Volvo Cars. The aim of the analysis is to continuously inspect changes in the AUTOSAR meta-model during the development of one AUTOSAR release, in order to be able to influence their standardization. For example, if an accepted change in the AUTOSAR meta-model removes one meta-element that is used at Volvo Cars, the change should be re-discussed in the AUTOSAR consortium involving representatives from Volvo Cars.

The *SREA* tool implements both the measures and the method defined in Papers G and H related to monitoring of the evolution of system design requirements and their specification. The tool is primarily used in one of the following two scenarios:

1. **To facilitate analysis of differences between requirements specifications from different AUTOSAR releases**

   The analysis is primarily done by the AUTOSAR team at Volvo Cars with support from other teams (e.g., system architects and designers). The aim of the analysis is to reduce time for analyzing differences between requirement specifications from different AUTOSAR releases.

2. **To facilitate analysis of differences between requirements specifications related to specific AUTOSAR features**

   The analysis is primarily done by the AUTOSAR team at Volvo Cars with support from other teams (e.g., system architects and designers). The aim of the analysis is to reduce time for analyzing differences between requirement specifications related to a specific new AUTOSAR feature. This analysis is meant to complement the analysis performed by the *ARCA* tool explained in the second bullet. For example, *SREA*

tool can list all introduced, removed and/or modified requirements related to a set of new modeling elements introduced to the AUTOSAR meta-model by the analyzed feature.

In addition to their use at Volvo Cars, there are three additional types of industrial application of these three tools:

1. They can be used without modifications by other car manufacturers and their software/tool suppliers in the similar scenarios as explained above, as long as they follow the AUTOSAR standard in the development.

2. The *ARCA* tool can be used by the AUTOSAR consortium to generate AUTOSAR meta-model change documentation, and the *SREA* tool can be used to generate change documentation of the AUTOSAR requirements specifications, related to different AUTOSAR features. Together, *ARCA* and *SREA* can also be used to indicate which meta-model elements (syntax) require further explanations in the specifications (semantics) related to their use in the AUTOSAR models.

3. Finally, all three tools can be used, with certain modifications, by the companies from other domains who work according to the principles of MDE. On the one side, they would need to re-implement the meta-model, model and requirements importers in the *ARCA*, *QTool*, and *SREA* tools, respectively, depending on the format they use for these artifacts. On the other hand, they would need to configure the tools in order to analyze relevant types of changes, e.g., meta-model packages and tagged values in case of *ARCA* and taxonomy of changes in case of *SREA*.

1.3.2 Individual contribution

In Papers A, B, C, E, F and H, the PhD candidate was the main contributor in planning and execution of the studies, and writing of the publications. In Paper D and G, the PhD candidate was the main contributor in planning of the studies, and participated in writing of the publications. The execution of the studies was performed by the first author of the papers, i.e., Maxime Jimenez and Corrado Motta, respectively.

The PhD student was the main contributor in writing of the "AUTOSAR" book chapter in the "Automotive Software Architectures" book.

The PhD student was the main contributor in the implementation of the *QTool* and *ARCA* tool, while Corrado Motta was the main contributor in the implementation of the *SREA* tool.

1.3.3 Related publications

As indicated in the list of publication under "Other publications", three of our publications related to the work presented in this thesis are not included in this thesis. In this subsection, we briefly describe their contribution and explain why they are not included in the thesis.

1. "**Measuring the Size of Changes in Automotive Software Systems and their Impact on Product Quality**" written by *D. Durisic,*

M. Staron and M. Nilsson and published in the *Proceedings of the 12th International Conference on Product Focused Software Development and Process Improvement (PROFES), pp. 10-13, 2011.*

This publication is entirely contained in Paper A that represents its extension by providing empirical validation, detailed explanation of the proposed method, and detailed description of the research methodology, related work and generalization of the presented results.

2. "**Evolution of Long-Term Industrial Meta-Models - An Automotive Case Study of AUTOSAR**", written by *D. Durisic, M. Staron, M. Tichy, J. Hansson* and published in *Proceedings of the 40th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), pp. 141-148, 2014.*

   The paper presents the study which defines and uses seven software measures for analyzing and visualizing the evolution of the AUTOSAR meta-model related to different design roles in the automotive development process. The results of five of these measures (*Number of elements*, *Number of attributes*, *Number of changes*, *Number of changed elements* and *Number of changed attributes*) including the definition of design roles are already presented in the similar scope in Papers C, D, E and F. The remaining two measures (*Complexity* and *Average depth of inheritance*) are not considered relevant for answering the main research question of this thesis presented above.

3. "**Quantifying Long-Term Evolution of Industrial Meta-Models - A Case Study**" written by *D. Durisic, M. Staron, M. Tichy, J. Hansson* and published in the *Proceedings of the International Conference on Software Process and Product Measurement (MENSURA), pp. 104-113, 2014.*

   The paper presents the study which analyzes which of the five properties of the AUTOSAR meta-model, namely size, length, complexity, coupling and cohesion, are mostly affected by the AUTOSAR meta-model evolution. The analysis is done by calculating ten software measures between available historical releases of the AUTOSAR meta-model. The results of the measures and general conclusions of the paper are not considered relevant for answering the main research question of this thesis.

The PhD candidate was the main contributor in planning and execution of all three studies, and writing of all three publications.

1.4 Research methodology

The research methodology used in this thesis mainly consists of a series of case studies which aim to increase understanding of meta-models, models and system requirements, and validate our methods used for monitoring the evolution of certain properties of these three artifacts. Using only one research method, however, does not usually suffice for the combined work of practitioners and researchers for solving industrial problems [29]. Therefore,

our methods are developed following the methodology of constructive research which relies on the results of our case studies.

As all of our methods are based on the use of software measurement, we used the methodology of the GQM approach described in Section 1.2.1 as part of the constructive studies for developing the actual measures. In particular, we first defined the general goal for the measurement and the questions the measurement results need to answer about the analyzed entities for achieving this goal. We then developed the measures using conceptual models of these entities and their properties, e.g., domain-specific models and meta-models.

In this section, we first summarize the theory of case study and constructive research in Sections 1.4.1 and 1.4.2, respectively. We emphasize in bold the important terms used in Section 1.4.3 in which we demonstrate how we used the two research methods in our studies. Finally, we discuss threats to validity of our results in Section 1.4.4.

1.4.1 Case study theory

Case study is classified as an empirical research method [17] and it focuses on the examination of a real-world situation, which makes it suitable for industrial evaluations. Yin [43] defines a case study as an iterative process consisting of five phases, as shown in Figure Figure 1.5:

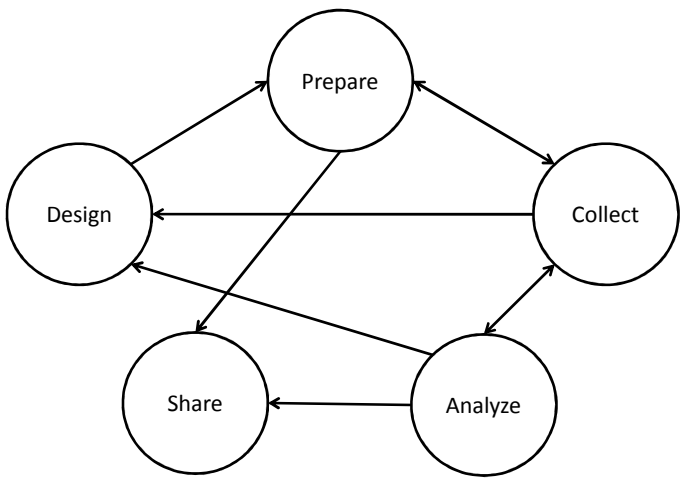


Figure 1.5: The five phases of a case study [43]

In this section, we focus on the design, collect and analyze phases. The design phase consists of the following five components [43]: (i) Research questions, (ii) Study propositions, (iii) Unit of analysis, (iv) Linking data to propositions and (v) Criteria for interpreting the findings.

The design of each study begins with a clear definition of **research questions**. Providing answers to these questions is the main goal of the case study. In order to understand how to achieve this goal, however, the scope of the study is defined together with the identification of elements which are to be examined (the study propositions). Next, the **unit of analysis** (i.e., the "case"), the elements of which are to be examined, is defined. The data obtained from this

examination are then linked to the propositions in order to answer the defined research questions using, e.g.,, statistical analysis. Finally, the criteria for data interpretation are defined in order to indicate when the obtained results can be considered valid, e.g., by defining the statistical significance.

The **data collection phase** can include both qualitative and quantitative data collection methods [17]. Qualitative data can be obtained by analyzing documentation, performing observations, conducting interviews, etc. Interviews are especially common in analyzing industrial cases because they provide quick answers to question from experts. They can be formal with a precisely-defined set of questions, informal relying on a casual discussion with experts [35] or semi-formal where questions are pre-defined, but can be deviated from during the interview [7]. Quantitative research represents the analysis of numerical data in order to explain a certain phenomenon [1]. Quantitative data is usually obtained by measurement.

In the **data analysis phase**, the data can be analyzed using different methods, e.g., pattern matching (comparing the empirical pattern with one or several predicted patterns) and explanation building (e.g., using theoretically proven concepts) [43]. Quantitative data can be analyzed using a number of statistical methods, including correlation and time-series analysis.

1.4.2 Constructive research theory

"The aim of constructive research is to solve practical problems while producing an academically appreciated theoretical contribution" [25]. Constructive studies consist of the following six main phases [21]:

1. Understanding the industrial context and selecting a problem (case study)
2. Understanding the study area (literature review)
3. Designing one or more solutions (creation of a novel construct)
4. Demonstrating the solution's feasibility (case study)
5. Validating the results (case study)
6. Generalizing the results (possible further case studies)

Understanding the industrial context and selecting a problem usually involves a case study which aims to increase the understanding of the industrial context and identify relevant problems for further studies, e.g., by conducting interviews and documentation reviews. **Understanding the study area** aims to increase the understanding of the scientific problem and the surrounding theory, usually by conducting literature reviews. **Designing one or more solutions** aims to solve the identified industrial and scientific problem by creating a novel construct, e.g., a method, tool, technique or process [27], and it is based on the knowledge obtained in the first two phases.

The phases of **demonstrating the solution's feasibility** and **validating the results** usually require additional case studies, in which the novel construct is evaluated and empirically validated on the chosen units of analysis. Finally, the phase of **generalizing the results** involves a detailed discussion

on the practical use of the novel construct in other industrial contexts that face similar problems, and may therefore involve further case studies.

As we can see from this description, all phases of constructive research, except from the second phase, usually require conducting a case study. This is why constructive research and case study methods should be combined for solving practical problems and delivering academically rigorous contribution. The benefit of this combined approach is that solutions for the relevant practical problems are produced quickly and their feasibility is directly tested in the real industrial contexts. The use of constructive research with case studies has some drawbacks as well. One drawback is that multiple case studies are usually needed for generalizing the results. Another drawback that the produced solutions may not always be the best and/or optimal solutions, which could have been identified by, e.g., conducting longer experiments with different parameters and testing them in multiple scenarios.

1.4.3 Research methods used in our papers

In order to address our research questions, we conducted eight studies, each described in one of the Papers A-H. Three studies were conducted using the case study method, four studies were conducted using constructive research which included additional case studies in the way described in the previous section, and one study was a tool presentation. The description of the research methodology we used in each study/paper is described below.

1. **Paper A**

   The study presented in Paper A was conducted following the constructive research method. In order to understand the industrial context and select a problem for the study, we first conducted a case study using architectural models from Volvo Cars as units of analysis. We analyzed the organization of these architectural models. As the outcome, we formulated our study problem in *RQ1*. In order to understand the study area, we performed a literature review of the complexity measures that can be used for monitoring the evolution of architectural models.

   We used the outcomes of the case study and the literature review to design a solution to the problem defined in *RQ1*. The solution represents a new method based on two software measures developed following the GQM approach. The goal of the measurement was to monitor the complexity increase of the automotive architectural models when new functionality is introduced into the system. In order to achieve this goal, we searched for the answer to the questions of what is the complexity and coupling change between two version of one architectural model resulting in the definition of the complexity and the coupling measures. We also developed the *QTool* which is able to calculate the two measures for the two evolving architectural models at different time points.

   In order to demonstrate the solution's feasibility and validate the results empirically, we conducted another case study in which we evaluated our method and the *QTool* using two architectural models from Volvo Cars as units of analysis. The results are empirically validated by comparing

the measurement results with the expectations from the experts from Volvo Cars on the complexity increase of the two analyzed models.

2. **Paper B**

   The study presented in Paper B was conducted following the case study method using AUTOSAR meta-model as a unit of analysis. The study aimed to increase the understanding of the industrial context and select a relevant problem for the constructive research presented in Paper C. In the design phase, we defined *RQ2* - *RQ4*. In the data analysis phase, we studied the organization of the AUTOSAR meta-model and created examples using four meta-modeling approaches in order to address *RQ2* and *RQ3*. In order to address *RQ4*, we identified that the evolution of domain-specific meta-models related to different architectural features and its impact on the development projects is one of the biggest challenges for the meta-modeling practitioners. We used this finding to formulate our research problem in Paper C.

3. **Paper C**

   The study presented in Paper C was conducted following the constructive research method. In Paper B, we already identified that the evolution of domain-specific meta-models related to different architectural features and its impact on the development projects is one of the biggest practical meta-modeling problems. We used this finding to formulate our general study problem in *RQ5*, and divided it into more concrete study problems formulated in *RQ6* - *RQ8*. In order to understand the industrial context, we extended the knowledge obtained in Paper B related to the organization of the AUTOSAR meta-model by conducting an interview with one of the AUTOSAR's change managers. This helped us to to understand how changes related to different AUTOSAR features are implemented in the AUTOSAR meta-model. In order to understand our study area, we performed a literature review of the meta-model differentiation techniques and applicable measures.

   We used the outcomes from Paper B, the interview and the literature review to design a solution to the problems defined in *RQ6* and *RQ7*. The solution represent two constructs: (i) a data model that is able to capture relevant changes between different AUTOSAR meta-model versions (*RQ6*), and (ii) a measure of meta-model change (*NoC*) which is developed following the GQM approach based on this data-model. The goal of the measurement was to monitor changes between different meta-model versions. In order to achieve this goal, we searched for the answer to the question of how many atomic changed occurred between two meta-model versions resulting in the definition of the *NoC* measure. We also developed the *ARCA* tool that is able to calculate *NoC* between two AUTOSAR meta-model releases, related to a set of chosen AUTOSAR features, and targeting a set of chosen AUTOSAR modeling tools.

   In order to demonstrate the solution's feasibility and validate the results empirically, we conducted a case study in which we evaluated and validated the data model and the *NoC* measure using AUTOSAR features and AUTOSAR modeling tools as unit of analysis. We asked the experts

from different tooling vendors to provide us with the estimated efforts needed to support different AUTOSAR features in their modeling tools. We used this collected data in the data analysis phase to perform correlation analysis between the results of *NoC* and the estimated efforts in order to address *RQ8*. The generalization of our findings from this study is described in Paper D.

4. **Paper D**

   The study presented in Paper D was conducted following the case study method using Modelica and UML meta-models as units of analysis. The study aimed to generalize our findings described in Paper C, i.e., to assess the applicability of our data model and the *NoC* measure on additional meta-models. In the design phase, we defined *RQ9*. In the data analysis phase, we performed a study of the organization of Modelica and UML meta-models. We calculated *NoC* between different releases of these two meta-models, respectively, in order to address *RQ9*.

5. **Paper E**

   The study presented in Paper E was conducted following the constructive research method. The study aimed to extend the results presented in Paper C so that they can be used for identifying the optimal set of architectural features to be used in the development projects. Therefore, we formulated our study problems in *RQ10* and *RQ11*. In order to understand the study area, we performed a literature review of the existing optimization methods.

   We used the *NoC* measure presented in Paper C and the literature review to design a solution to the problems defined in *RQ10* and *RQ11*. The solution involved constructing a method (*MeFiA*) that is able to calculate the *NoC* measure in the domain-specific meta-models related to different architectural features (*RQ10*). The method is also able to identify the optimal set of features to be adapted in the development projects based on their meta-model impact (*RQ11*). We also extended the *ARCA* tool to be able to identify this optimal set in car development projects.

   In order to demonstrate the solution's feasibility, we conducted a case study in which we evaluated the *MeFIA* method using AUTOSAR meta-model as a unit of analysis.

6. **Paper F**

   The study presented in Paper H describes the *ARCA* tool which is developed in the constructive studies described in Papers C and E. The study also shows the industrial use of *ARCA* in order to address *RQ12*. The presented data is obtained from the two case studies described in Papers C and E, which were used to demonstrate the feasibility of the solutions developed in the constructive studies of these two papers, respectively.

7. **Paper G**

   The study presented in Paper G was conducted following the constructive research method. In order to understand the industrial context and select a problem for the study, we first conducted a case study using using the

AUTOSAR standardized requirements as units of analysis. We analyzed the organization of the AUTOSAR requirement specifications and the structure of the requirements itself. As the outcome, we formulated our study problem in *RQ13*. In order to understand the study area, we performed a literature review of the requirements engineering field focusing on how to cope with the evolution of system requirements.

We used the outcomes of the case study and the literature review to design a solution to the problem defined in *RQ13*. The solution involved constructing a method that is able to (i) analyze the evolution of requirements related to different specifications and (ii) calculate a number of measures related to different specification. The measures are developed following the GQM approach. The goal of the measurement was to monitor changes in the specifications between their different releases. In order to achieve this goal, we searched for the answer to the questions of (i) how many requirements are semantically changed between the two versions of one specification and (ii) how mature is the new version. This resulted in the definition of a set of measures for counting added, deleted and modified requirements (inspired by the similar measures for added, deleted and modified meta-model elements), and the *RMI* measure. We also developed the *SREA* tool that is able to calculate the measures used by our method related to these specifications/requirements.

In order to demonstrate the solution's feasibility, we conducted another case study in which we evaluated our method and the *SREA* tool using a set of AUTOSAR specifications and their requirements.

8. **Paper H**

   The study presented in Paper H was conducted following the case study method using the AUTOSAR meta-modeling environment as a unit of analysis. In the design phase, we defined *RQ14*. In the data collection phase, we calculated the *NoC* (defined in Paper C) in the AUTOSAR meta-model (syntax) related to a set of chosen AUTOSAR features using the *ARCA* tool (presented in Paper F). We also calculated the *Number of changed requirements* (defined in Paper G) in the AUTOSAR specifications (semantics) related to these features using the *SREA* tool. In the data analysis phase, we calculated correlation between the results of the *ARCA* tool and *SREA* tools in order to address *RQ14*.

1.4.4 Research validity

According to Cook and Campbell [13], four types of validity threats to empirical studies conducted in the area of software engineering shall be considered. We explain below how we addressed each of them in our studies.

1. **Internal validity**

   Internal validity is concerned with the results of the analysis not being casual, i.e., the relationship between the measured properties and the outcome should not be random. The most severe threat to the internal validity in our studies was related to the measurement process which was performed by developing software tools, e.g., *QTool*, *ARCA* and *SREA*,

for calculating different measures, e.g., *NoC* and *RMI*. In order to ensure internal validity, we performed detailed testing of the tools using smaller examples before employing them for the measurements in the studies.

2. **External validity**

   External validity is concerned with the generalization of results. In our studies, this is related to the applicability of our results to models, meta-models and system requirements used by other companies facing similar problems. There are two particular threats to the external validity of our studies. The first threat is that the proposed methods and tools would apply only to the automotive software development process at Volvo Cars and not to other automotive companies. In order to minimize this threat, we included other automotive companies (OEMs and software/tool suppliers from the AUTOSAR consortium) in the process of defining and evaluating our methods and tools.

   The second threat is related to the AUTOSAR meta-modeling environment (AUTOSAR meta-model and AUTOSAR requirements) that was the unit of analysis in Papers A - C and E - H . The proposed methods and tools we applied to the AUTOSAR meta-modeling environment should also be applicable to other domain-specific meta-models. Therefore, we mapped the layers of the AUTOSAR modeling environment to the layers of MOF which is a commonly accepted modeling hierarchy. In addition to this, we discussed in the included papers the steps that need to be taken in order to apply the proposed methods to meta-models of other domains, e.g., avionics and telecommunications.

   Finally, the goal of Paper D was to minimize the external validity threat of the *NoC* measure, which is used in Papers C, E, F and H. Therefore, we calculated the measure to two additional meta-models of Modelica and UML. As opposed to the AUTOSAR meta-model which is an architectural meta-model, UML is a general-purpose meta-model and Modelica is a behavioral meta-model. Therefore, we analyzed three different types of meta-models which significantly reduces the threat that *NoC* is only applicable for measuring the evolution of the AUTOSAR meta-model.

3. **Construct validity**

   Construct validity is concerned with the mismatch between theory and observations. In our studies, this was related to the ability of the measures to capture the desired properties of the analyzed system. In order to define our measures in a rigorous way, all measures were defined according to the GQM approach based on the data model that enabled us to have open discussions about the ability of the measures to capture the desired properties of the measured entities. Additionally, we performed theoretical and empirical validation of the complexity and coupling measures presented in Paper A. We also performed empirical validation of the *NoC* measure used in Papers C, E, F and H. Finally, we showed that it is possible to calculate the measures of requirements evolution used in Papers F and H on a case of AUTOSAR requirement specifications.

4. **Conclusion validity**

   Conclusion validity is concerned with the degree to which the conclusions of the studies are reasonable. In Papers C and H, this was related to the significance of the results from the statistical analysis, which was high. In Papers A, D, E, F and G, the conclusions were derived based on studying and applying our methods to industrial scenarios. The conclusion was that the results could capture the desired properties, thus validating our hypothesis that simple measures can be used as preliminary indicators. Finally, Paper B presents our opinion based on our experience in working with domain-specific meta-models, and shall be taken as a position paper.

1.5 Conclusions and future work

The main contribution of this thesis are three methods (and software tools) that can be used for automated impact assessment of using new architectural features, that require updates of the modeling language, on the development projects. We showed that using these methods and combining their results in the development process is able to accelerate the work of system designers responsible for planning architectural updates in the development of automotive software systems. This, in turn, enables faster and cheaper innovation cycles in the car development projects.

The first method and the tool (*QTool*) are based on two structural measures of complexity and coupling in the architectural models. The method is able to identify parts of the system which became overly complex after the implementation of new architectural features. We showed that this information can be used by the automotive system designers to identify and prioritize testing areas in the system and/or rework the system's architecture in order to reduce the number of potential faults. This, in turn, contributes to the increased speed of automotive software development and higher quality of software.

The second method (*MeFIA*) and the tool (*ARCA*) are based on the measure of change in the architectural meta-models (NoC). The method is able to estimate the impact of using new architectural features on the used modeling tools. We showed that this information can be used by the automotive system designers to decide which architectural features shall be used in the system, and to plan updates of the modeling tool-chain.

Additionally, we showed that with the help of the *ARCA* tool, it is possible to identify the actual AUTOSAR meta-model changes caused by a particular feature and relevant to a specific modeling tool used by one design role. This, in turn, has a potential to significantly reduce the number of inspected AUTOSAR meta-model changes in the detailed impact assessment for different modeling tools, e.g., from more than 35000 changes to less than a hundred for the majority of AUTOSAR features and tools.

The third method and the tool (*SREA*) are based on the measure of change in the system design requirements. The method is able to identify the subset of design requirements which are affected by the use of new architectural features. We showed that this information can be used by the automotive system designers to more quickly understand how to use the new features in the architectural models, and to identify unstable requirement specification which

are candidates for inspection. This, in turn, has a potential to significantly reduce the number of analyzed design requirements, e.g., from more than 20000 requirements to less than a hundred for the majority of AUTOSAR features.

In order to achieve the full benefit of using these methods and tools in the development process, they should be used in combination. *ARCA* and *SREA* tools shall be used before one architectural feature is supported in the development process in two different ways: First, to indicate whether it is feasible to use the feature in the project time-frame considering its impact on the used modeling tools (*ARCA*) and requirements specifications (*SREA*). Second, to plan updates of the tools (*ARCA*) and analysis of the changes in the requirements specifications (*SREA*), in case it is considered feasible. The *QTool* shall be used after one feature is supported in the development process and used in the system models in order to indicate which parts of the system are mostly affected by the new feature. This information can be used to steer testing activities and possible restructuring of the system's architecture in case of significant complexity increase of its components.

As we can see, all three methods and tools are based on quantitative analysis of changes using one or more simple software measures. This validates our hypothesis that quantitative analysis of evolution of the three main MDE artifacts related to different architectural features can serve as a valuable early indicator of which features shall be used in the system, and what is their impact on the development projects.

The results presented in this theses provide opportunities for further research in the area of meta-model and system requirements evolution. Related to the evolution of meta-models, one direction could be to classify meta-model changes into categories according to their impact on different segments of the modeling tools (e.g., graphical user interface, tool importers or underlying data-base) and identify the ones that require most rework in the tools. The results of such a study can then be used to extend the *MeFIA* method and the *ARCA* tool to consider these categories in the measurement process e.g, by performing category based measurements or assigning different weights to different categories. This, in turn, can be used to steer the development of the tools where certain changes could probably be done automatically, e.g., changes affecting databases of the tools or simple textual editors of the meta-modeling elements and their properties.

Another direction could be to analyze the relation between different features based on their impact on the same parts of the meta-model. The results of such a study can be used to group similar features that should be used together in the development project, e.g., at the cost of tool-chain update that is much lower than the sum of costs of supporting each feature separately duo to the overlap of changes. This, in turn, can be used to complement the results of the *MeFIA* method in the analysis of which architectural features shall be used in the system.

Related to the evolution of system requirements, one direction could be to include natural language processing (NLP) techniques in the analysis performed by the *SREA* tool. This approach has a potential to additionally increase the speed of analyzing changes in the requirements specifications related to the use of new architectural features in the system, by filtering out syntactically changed requirements that have no semantic impact.

Bibliography

[1] M. Aliaga and B. Gunderson. *Interactive Statistics, end edition*. Prentice Hall, 1999.

[2] U. Aßmann, S. Zschaler, and G. Wagner. "Ontologies, Meta-models, and the Model-Driven Paradigm". In: *Ontologies for Software Engineering and Software Technology*. Springer Berlin Heidelberg, 2006, pp. 249–273.

[3] C. Atkinson and T. Kühne. "Model-Driven Development: A Metamodeling Foundation". In: *Journal of IEEE Software* 20.5 (2003), pp. 36–41.

[4] C. Atkinson and T. Kühne. "Strict Profiles: Why and How". In: *In Proceedings of the 3rd International Conference on the Unified Modeling Language, Lecture Notes in Computer Science*. 2000, pp. 309–322.

[5] C. Atkinson, T. Kühne, and B Henderson-Sellers. "To Meta or not to Meta - That is the Question". In: *Journal of Object - Oriented Programming* 13.8 (2000), pp. 32–36.

[6] *Automotive Open System Architecture*. AUTOSAR. 2003. URL: `www.autosar.org`.

[7] L. Barriball and A. While. "Collecting Data Using a Semi-Structured Interview: A Discussion Paper". In: *Journal of Advanced Nursing* 19.2 (1994), pp. 328–335.

[8] V. Basili, G. Caldiera, and H. Rombach. *The Goal Question Metric Approach*. Encyclopedia of Software Engineering, Wiley, 1994.

[9] Jean Bézivin and Olivier Gerbé. "Towards a Precise Definition of the OMG/MDA Framework". In: *International Conference on Automated Software Engineering*. 2001, pp. 273–280.

[10] X. Blanc, J Delatour, and T. Ziadi1. "Benefits of the MDE Approach for the Development of Embedded and Robotic Systems". In: *Proceedings of the Workshop on Control Architectures of Robots: from Models to Execution on Distributed Control Architectures*. 2007.

[11] L. Briand, K. El Emam, and S. Morasca. *Theoretical and Empirical Validation of Software Product Measures*. Tech. rep. International software Engineering Research Network, 1995.

[12] L.C. Briand, S. Morasca, and V.R. Basili. "Property-based Software Engineering Measurement". In: *IEEE Transactions on Software Engineering* 22.1 (1996), pp. 68–86.

[13] T. Cook and D. Campbell. *Quasi-Experimentation: Design & Analysis Issues for Field Settings*. Houghton Mifflin, 1979.

[14] D Di Ruscio, L. Iovino, and A. Pierantonio. "Evolutionary Togetherness: How to Manage Coupled Evolution in Metamodeling Ecosystems". In: *Proceedings of the 6th International Conference on Graph Transformations*. 2012, pp. 20–37.

[15] N. E. Fenton and S. L. Pfleeger. *Software Metrics: A Rigorous and Practical Approach, 2nd edition*. London, International Thomson Computer Press, 1996.

[16] F. García et al. "Towards a Consistent Terminology for Software Measurement". In: *Journal of Information and Software Technology* 48.8 (2006).

[17] R. Glass. "The Software Research Crisis". In: *Journal of IEEE Software* 11.6 (1994), pp. 42–47.

[18] M. Hiller. *Surviving in an Increasingly Computerized and Software Driven Automotive Industry*. Keynote speach at the International Conference on Software Architectures. 2017. URL: `https://www.youtube.com/watch?v=mpbWQbkl8_g`.

[19] *Information Technology - Information Resource Dictionary System*. ISO / IEC 10027. 1990.

[20] M. Jørgensen. "Software Quality Measurement". In: *Journal of Advances in Engineering Software* 30.12 (1999), pp. 907–912.

[21] E. Kasanen and A. Siitonen. "The Constructive Approach in Management Accounting Research". In: *Journal of Management Accounting Research* 5 (1993), pp. 241–264.

[22] S. Kent. "Model Driven Engineering". In: *Proceedings of the International Conference on Integrated Formal Methods*. 2002, pp. 286–299.

[23] B. Kitchenham, S. L. Pfleeger, and N. Fenton. "Towards a Framework for Software Measurement Validation". In: *Journal of IEEE Transactions on Software Engineering* 21.12 (1995), pp. 929–944.

[24] A. Kleppe. "A Language Description is More than a Metamodel". In: *Proceedings of the 4th International Workshop on Software Language Engineering*. 2007, pp. 273–280.

[25] L. Lehtiranta et al. "The Constructive Research Approach: Problem Solving for Complex Projects". In: *Designs, Methods and Practices for Research of Project Management*. Gower, 2015. Chap. 8, pp. 95–106.

[26] P. Liggesmeyer and M. Trapp. "Trends in Embedded Software Engineering". In: *Journal of IEEE Software* 26.3 (2009), pp. 19–25.

[27] K. Lukka and T. Reponen. "The Key Issues of Applying the Constructive Approach to Field Research". In: *Management Expertise for the New Millennium: In Commemoration of the 50th Anniversary of the Turku School of Economics and Business Administration* (2000), pp. 113–128.

[28] F. Mantz, G. Taentzer, and Y. Lamo. "Well-formed Model Co-evolution with Customizable Model Migration". In: *Proceedings of the International Workshop on Principles of Software Evolution*. 2013, pp. 1–10.

[29] L. Mathiassen. “Collaborative Practice Research”. In: *Journal of Information Technology & People* 15.4 (2002), pp. 321–345.

[30] *MOF 2.0 Core Specification*. Object Management Group. 2004. URL: www.omg.org.

[31] G. Nordstrom et al. “Metamodeling - Rapid Design and Evolution of Domain-Specific Modeling Environments”. In: *IEEE Conference on Engineering of Computer Based Systems*. 1999, pp. 68–74.

[32] *Object Constraint Language Version 2.0*. Object Management Group. 2006. URL: www.omg.org.

[33] *Object Management Group*. OMG. 1989. URL: www.omg.org.

[34] *Open Modelica User Guide (Releases)*. OpenModelica. 2016. URL: www.openmodelica.org.

[35] C. Robson. *Real World Research: A Resource for Social Scientists and Practitioner-Researchers, 2nd edition*. Blackwell Oxford, 2002.

[36] J. Rosenberg. “Some Misconceptions about Lines of Code”. In: *Proceedings of the 4th International IEEE Symposium on Software Metrics*. 1997, pp. 137–142.

[37] M. Saeki and H. Kaiya. “On Relationships among Models, Meta Models and Ontologies”. In: *6th OOPSLA Workshop on Domain-Specific Modeling*. 2007.

[38] *Standard for a Software Quality Metrics Methodology*. IEEE. 1998.

[39] *Systems and Software Engineering - Measurement Process*. ISO/IEC 15939. 2007.

[40] *Systems and Software Engineering - Product Quality*. ISO/IEC 9126-1. 1991.

[41] R. Taylor, N. Medvidovic, and E. Dashofy. *Software Architecture: Foundations, Theory, and Practice*. Wiley, 2009.

[42] *Unified Modeling Language 2.5*. Object Management Group. 2015. URL: www.omg.org.

[43] R. Yin. *Case Study Research: Design and Methods, 5th edition*. London, SAGE, 2014.

Chapter 2

Case Study Background

Automotive Software Development based on AUTOSAR

D. Durisic

Based on Chapter 4 of the Automotive Software Architectures book, Springer, 2017

Abstract

In this chapter, we describe the role of AUTOSAR (AUTomotive Open System ARchitecture) standard in the development of automotive system architectures. AUTOSAR defines the reference architecture and methodology for the development of automotive software systems, and provides the language (meta-model) for their architectural models. It also specifies the architectural modules and functionality of the middleware layer known as the *basic software*. We start by describing the layers of the AUTOSAR reference architecture. We then describe the proposed development methodology by identifying major roles in the automotive development process and the artifacts they produce including an example of each artifact. We follow up by explaining the role of AUTOSAR meta-model in the development process and show examples of the architectural models that instantiate it. Finally, we conclude by explaining the use of the AUTOSAR meta-model for configuring basic software modules.

2.1 Introduction

The architecture of automotive software systems, as software-intensive systems, can be seen from different views, as described by Kruchten in the *4+1 architectural view model* [9]. Two of these architectural views deserve special attention in this thesis, namely the logical and the physical views, so we describe them briefly here as well.

Logical architecture of the automotive software systems is responsible for defining and structuring high level vehicle functionalities such as auto-breaking when a pedestrian is detected on the vehicle's trajectory. These functionalities are usually realized by a number of logical software components, e.g., the *PedestrianSensor* component detects a pedestrian and requests full auto-break from the *BreakControl* component. These components communicate by exchanging information, e.g., about the pedestrian detected in front of the vehicle. Based on the type of functionalities they realize, logical software components are usually grouped into subsystems that in turn are grouped into logical domains, e.g., active safety and powertrain.

Physical architecture of the automotive software systems is usually distributed over a number of computers (today usually more than 100) referred to as Electronic Control Units (ECUs). ECUs are connected via electronic buses of different types (e.g., Can, FlexRay and Ethernet) and are responsible for executing one or several high level vehicle functionalities defined in the logical architecture. This is done by allocating logical software components responsible for realizing these functionalities to ECUs, thereby transforming them into runnable ECU application software components. Each logical software component is allocated to at least one ECU.

Apart from the physical system architecture that consists of a number of ECUs, each ECU has its own physical architecture that consists of the following main parts:

- Application software that consists of a number of allocated software components and is responsible for executing vehicle functionalities realized by this ECU, e.g., detecting pedestrians on the vehicle's trajectory.
- Middleware software responsible for providing services to the application software, e.g., transmission/reception of data on the electronic buses and tracking diagnostic errors.
- Hardware that includes a number of drivers responsible for controlling different hardware units, e.g, electronic buses and CPU of the ECU.

The development of the logical and physical architectural views of the automotive software systems and their ECUs is mostly done following the MDA (Model-Driven Architecture) approach [13]. This means that the logical and physical system architecture and the physical ECU architecture are described by means of architectural models. Looking into the automotive architectural design from the process point of view, car manufacturers (OEMs - Original Equipment Manufacturers) are commonly responsible for the logical and physical design of the system, while a hierarchy of suppliers are responsible for the physical design of specific ECUs, implementation of their application and middleware software and providing the necessary hardware [6].

In order to facilitate this distributed design and development of the automotive software systems and their architectural components, AUTOSAR (AUTomotive Open Systems ARchitecture) standard was introduced in 2003 as a joint partnership of automotive OEMs and their software and hardware vendors. Today, AUTOSAR consists of more than 150 global partners [2] and is therefore considered as *de facto* standard in the automotive domain. AUTOSAR is built upon the following major objectives:

1. Standardization of the reference ECU architecture and its layers. This increases the re-useability of the application software components in different car projects (within one or multiple OEMs) developed by the same software suppliers.

2. Standardization of the development methodology. This enables collaboration between different parties (OEMs and a hierarchy of suppliers) in the software development process for all ECUs in the system.

3. Standardization of the language (meta-model) for the architectural models of the system/ECUs. This enables a smooth exchange of architectural models between different modeling tools used by different parties in the development process.

4. Standardization of the ECU middleware (basic software - BSW) architecture and functionality. This allows engineers from OEMs to focus on the design and implementation of high level vehicle functionalities that can, in contrast to ECU middleware, create competitive advantage.

In the next four subsections, we show how AUTOSAR achieves each one of these objectives.

2.2 AUTOSAR reference architecture

The architectural design of ECU software based on AUTOSAR is done according to the 3-layer architecture that is build upon the ECU hardware layer, as presented in Figure 2.1.

The first layer, *Application software*, consists of a number of software components that realize a set of vehicle functionalities by exchanging data using interfaces defined on these components (referred to as ports). This layer is based on the logical architectural design of the system. The second layer, *Run-time environment* (RTE), controls the communication between software components abstracting the fact that they may be allocated onto the same or different ECUs. This layer is usually generated automatically based in the software component interfaces. If the software components are allocated onto different ECUs, transmission of the respective signals on the electronic buses is needed which is done by the third layer (*Basic software*).

Basic software layer consists of a number of BSW modules and it is responsible for the non-application related ECU functionalities. One of the most important basic software functionalities is the *Communication* between ECUs, i.e., signal exchange. It consists of BSW modules such as *COM* (Communication Manager) that is responsible for signal transmission and reception.

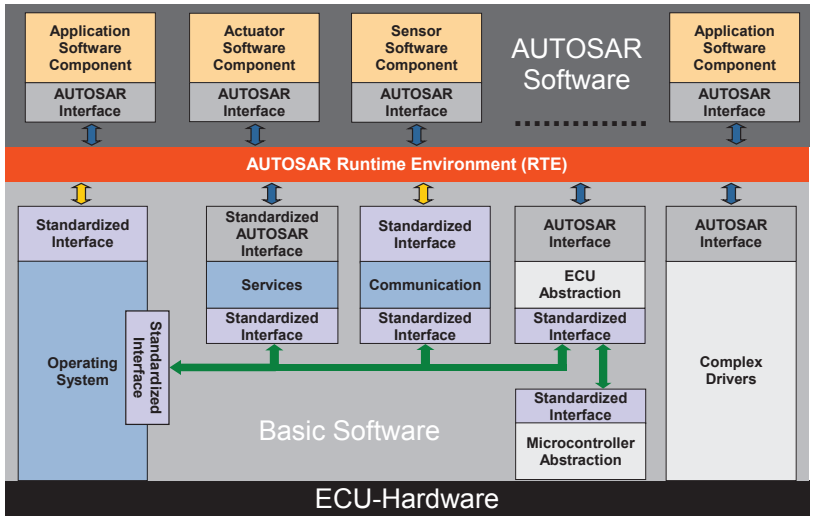


Figure 2.1: AUTOSAR layered software architecture [4]

However, AUTOSAR basic software also provides a number of *Services* to the *Application software* layer, e.g., diagnostics realized by *DEM* (Diagnostic Event Manager) and *DCM* (Diagnostic Communication Manager) modules that are responsible for logging error events and transmitting diagnostic messages, respectively, and the *Operating System* for scheduling ECU runnables. The majority of BSW modules are configured automatically based on the architectural models of the physical system [11], e.g., periodic transmission of a set of signals packed into frames on a specific electronic bus.

Communication between higher level functionalities of ECU *Basic software* and drivers controlling the ECU hardware realized by the *Microcontroler Abstraction* BSW modules is done by the *ECU Abstraction* BSW modules, e.g., bus interfaces modules such as *CanIf* that is responsible for the transmission of frames containing signals on the CAN bus. Finally, AUTOSAR provides the possibility for the application software components to communicate directly with hardware, thus bypassing the layers of the AUTOSAR software architecture, by means of custom implementations of *Complex Drivers*. This approach is, however, considered non-standardized.

Apart from the *Complex Drivers*, *RTE* and modules of the *Basic Software* layer are completely standardized by AUTOSAR, i.e., AUTOSAR provides detailed functional specifications for each module. This standardization, together with the clear distinction between the *Application software*, *RTE* and *Basic software* layers, allows ECU designers and developers to specifically focus on the realization of high level vehicle functionalities, i.e., without the need to think about the underlying middleware and hardware. Application software components and BSW modules are often developed by different suppliers who specialize in either one of these areas, as explained in more details in the following section.

2.3 AUTOSAR development methodology

On the highest level of abstraction, automotive vendors developing architectural components following the AUTOSAR methodology can be classified into one of the following major roles in the automotive development process:

- **OEM**: responsible for the logical and physical system design.
- **Tier1**: responsible for the physical ECU design and implementation of the software components allocated onto this ECU.
- **Tier2**: responsible for the implementation of the ECU basic software.
- **Tier3**: responsible for supplying ECU hardware, hardware drivers and corresponding compilers for building the ECU software.

In most cases, different roles represent different organizations/companies involved in the development process. For example, one car manufacturer plays the role of OEM, two software vendors play the roles of the Tier1 and Tier2, respectively, and one "silicon" vendor plays the role of Tier3. However in some cases, these roles can also be played by the same company, e.g., one car manufacturer plays the role of OEM and Tier1 by doing logical and physical system design, physical ECU design and implementation of the allocated software components (in-house development), or one software vendor plays the role of Tier1 and Tier2 by doing both implementation of the software components and BSW modules. The development process involving all roles and their tasks is presented in Figure 2.2.

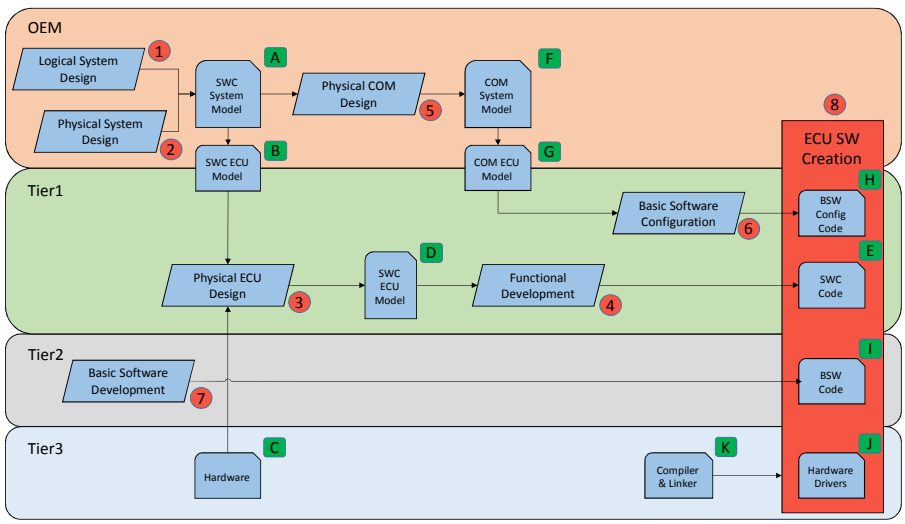


Figure 2.2: AUTOSAR development process

OEMs start with the *logical system design* (1) by modeling a number of composite logical software components and their port interfaces representing data exchange points. These components are usually grouped into subsystems

that are in turn grouped into logical domains. In the later stages of the development process, usually in the *physical ECU design* (3), composite software components are broken down into a number of atomic software components, but this could be done already in the logical system design phase by OEMs. An example of the logical system design of the minimalistic system created for the purpose of this chapter that calculates vehicle speed and presents its value to the driver is presented in Figure 2.3.

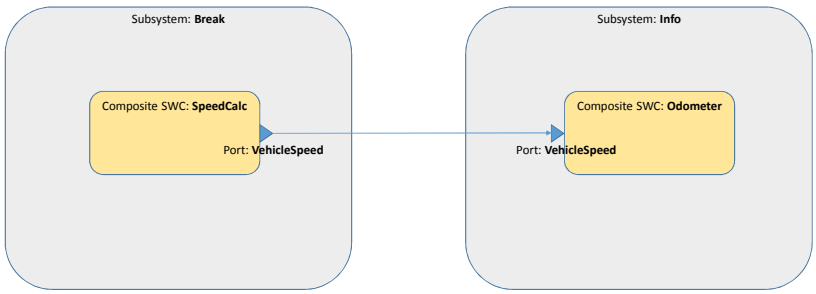


Figure 2.3: Example of the logical system design done by OEMs (1)

The example contains two subsystems, *Break* and *Info*, that each consists of one composite software component, *SpeedCalc* and *Odometer*, respectively. The *SpeedCalc* component is responsible for calculating vehicle speed and it provides this information via the *VehicleSpeed* sender port. The *Odometer* component is responsible for presenting the vehicle speed information to the driver and it requires this information via the *VehicleSpeed* receiver port.

As soon as certain number of subsystems and software components have been defined in the *logical system design* phase (1), OEMs can start with the *physical system design* (2) that involves modeling a number of ECUs connected using different electronic buses and deployment of the software components to these ECUs. In case two communicating software components (with connected ports) are allocated to different ECUs, this phase also involves the creation of the system signals that will be transmitted over the electronic bus connecting these two ECUs. An example of the physical system design of our minimalistic system is presented in Figure 2.4.

The example contains two ECUs, *BreakControl* and *DriverInfo*, connected using *Can1* bus. The *SpeedCalc* component is deployed to the *BreakControl* ECU and the *Odometer* component is deployed to the *DriverInfo* ECU. As these two components are deployed to different ECUs, they exchange information about the vehicle speed in a form of system signal named *VehicleSpeed*.

After the *physical system design* phase (2) is finished, detailed design of the car functionalities allocated to composite software components deployed to different ECUs (*physical ECU design*) can be performed by Tier1s (3). As different ECUs are usually developed by different Tier1s, OEMs are responsible for extracting the relevant information about the deployed software components from the generated *SWC system model* (A) into the *SWC ECU model* (B), known as the *ECU Extract*. The main goal of the physical ECU design phase is to break down the composite software components into a number of atomic software components that will in the end represent runnable entities

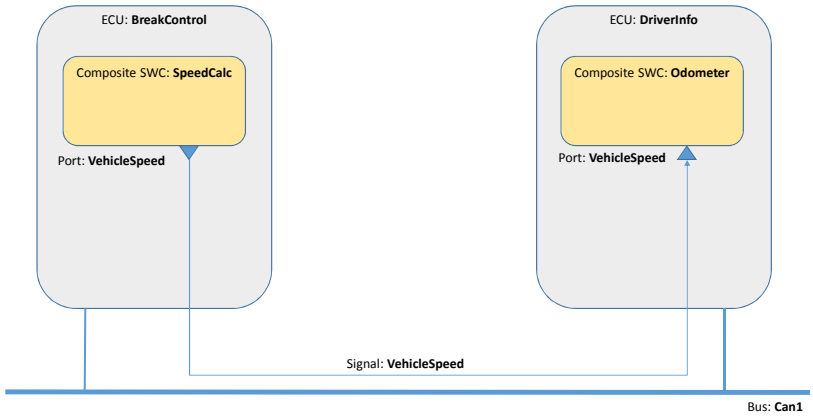
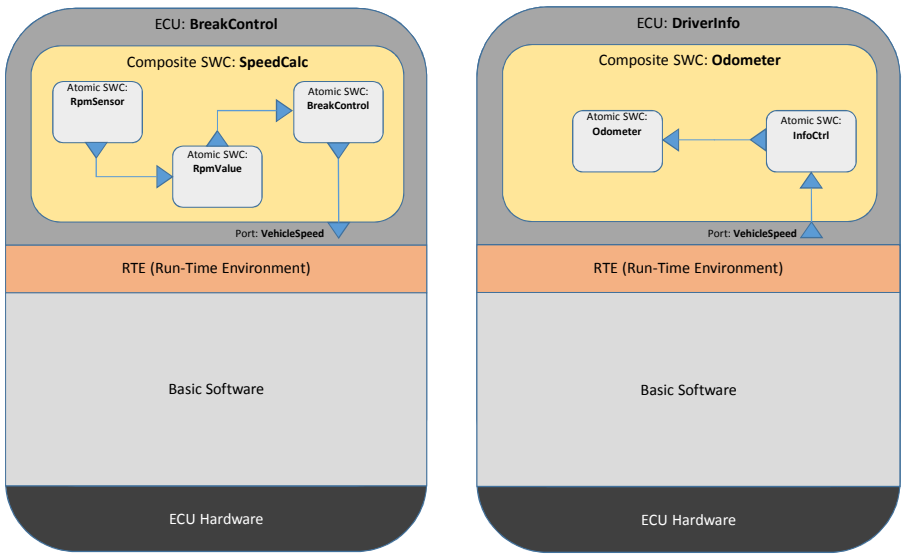


Figure 2.4: Example of the physical system design done by OEMs (2)

at ECU run-time. An example of the physical ECU design of our minimalistic system is presented in Figure 2.5.



(a) BreakControl ECU (b) DriverInfo ECU

Figure 2.5: Example of the physical ECU design done by Tier1s (3)

The example shows detailing of the *SpeedCalc* and *Odometer* composite software components into a number of atomic software components that will represent runnables in the final ECU software. *SpeedCalc* consists of the *RpmSensor* sensor component that measure the speed of axis rotation, the *RpmValue* atomic software component that calculates the value of the rotation and the *BreakControl* atomic software component that calculates the actual vehi-

cle speed based on the value of the axis rotation. *Odometer* consists of the *InfoControl* atomic software component that receives the information about the vehicle speed and the *Odometer* atomic software component that present the vehicle speed value to the driver.

The ECU design phase is also used to decide upon the concrete implementation data types used in the code for the data exchanged between software components based on the choice of the concrete ECU *hardware* (C) delivered by the Tier3s. For example, data can be stored as floats if the chosen CPU has a support for working with the floating points.

Based on the detailed *SWC ECU model* containing the atomic software components (D), Tier1s can continue with the *functional development* of the car functionalities (4) allocated onto these components. This is usually done with a help of behavioral modeling in the modeling tools such as Matlab Simulink, that are able to generate source *SWC code* for the atomic software components (E) automatically from the Simulink models [12]. This part is outside of the AUTOSAR scope.

During the physical ECU design and functional development phases performed by Tier1s, OEMs can work on the *physical COM design* (5) that aims to complete the system model with packing of signals into frames that are transmitted on the electronic buses. This phase is necessary for configuring the communication (COM) part of the AUTOSAR *basic software configuration* (6). An example of the physical COM design of our minimalistic system is presented in Figure 2.6.

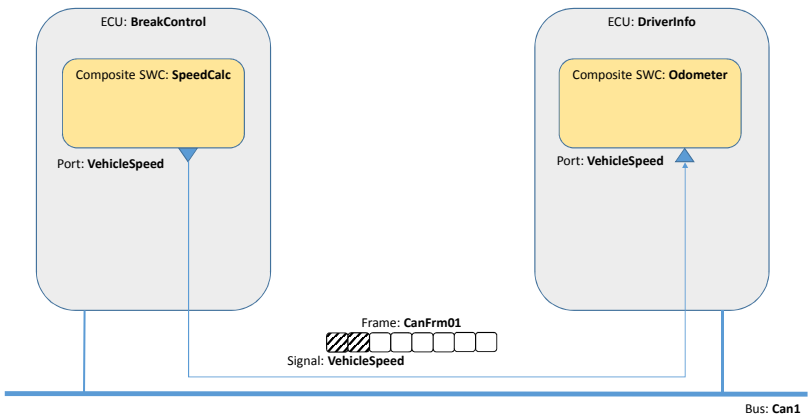
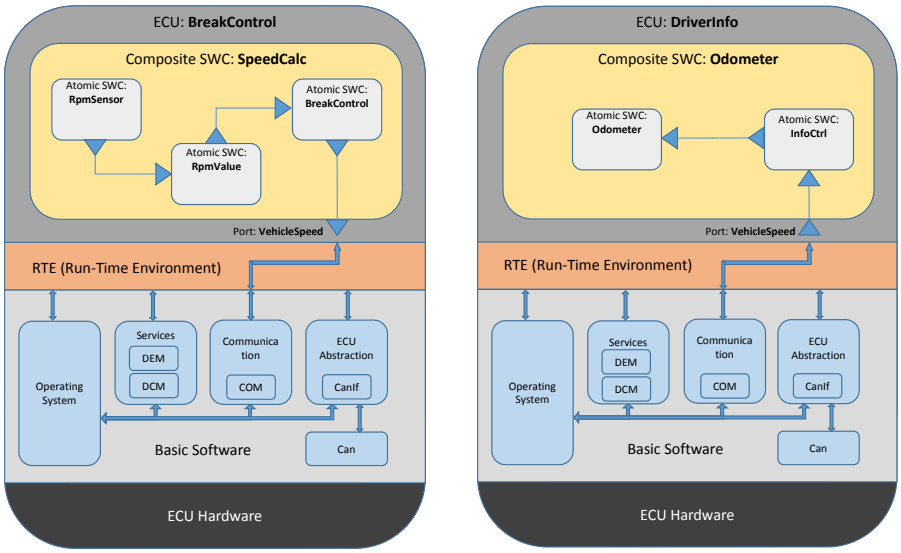


Figure 2.6: Example of the physical COM design done by OEMs (5)

The example shows one frame of eight bytes named *CanFrm01* that is transmitted by the *BreakControl* ECU on the *Can1* bus and received by the *DriverInfo* ECU. It transports the *VehicleSpeed* signal in its first two bytes.

After the physical COM design phase has been completed for the entire system, OEMs are responsible for creating *COM ECU model* extracts (G) from the generated *COM system model* (F) for each ECU that contains only ECU relevant information about the COM design. This step is similar to the step done after the logical and physical system design related to the extraction of the ECU relevant information about the application software components.



(a) BreakControl ECU (b) DriverInfo ECU

Figure 2.7: Example of the BSW configuration design done by Tier1s (6)

These ECU extracts are then sent to the Tier1s who use them as input for configuring the COM part of the ECU *basic software configuration* (6) and, together with configuring the rest of BSW (diagnostics services, operating sytem, etc.), generate the complete *BSW configuration code* (H) for the developed ECU. An example of the BSW configuration design of our minimalistic system is presented in Figure 2.7.

The example shows different groups of BSW modules, i.e., *Operating System*, *Services* including modules such as *DEM* and *DCM*, *Communication* including modules such as *COM*, and *ECU Abstraction* including modules such as *CanIf* needed for the transmission of frames on the Can bus in our example.

The actual ECU *basic software development* (7) is done by Tier2s based on the detailed specifications of each BSW module provided by the AUTOSAR standard, e.g., *COM*, *CanIf* or *DEM* modules. The outcome of this phase is a complete *BSW code* (I) for the entire basic software that is usually delivered by Tier2s in a form of libraries. The *hardware drivers* for the chosen hardware (J), in our example *CAN* driver, are delivered by Tier3s.

The last stage in the *ECU software creation* (8) is to compile and link the functional *SWC code* (E), *BSW configuration code* (H), functional *BSW code* (I) and the *hardware drivers* (J). This is usually done using the *compiler and linker* (K) delivered by Tier3s.

Despite the fact that the described methodology of AUTOSAR reminiscences of the traditional waterfall development approach, except from the decoupled development of the ECU functional code and the ECU BSW code, in practice it just represents one cycle of the entire development process. In other words, steps (1), (2), (3), (4), (5) and (6) are usually repeated a number

of times adding new functionalities to the system and its ECUs. For example, new composite software components are introduced in the logical system design (1) requiring new signals in the physical system design (2), new atomic software components are introduced as part of the new composite software components in the physical ECU design (3) and implemented in the functional development (4), and new frames to transport the new signals are introduced in the physical COM design (5) and configured in the BSW configuration design (6) phase. Sometimes even the ECU hardware (C) and its compiler/linker (K) and drivers (J) can be changed between different cycles, in case it cannot withstand the additional functionality.

2.4 AUTOSAR meta-model

As we have seen in the previous section, a number of architectural models, as outcomes of different phases in the development methodology, are exchanged between different roles in the development process. In order to assure that modeling tools used by OEMs in the logical (1), physical (2) and communication system design (5) phases are able to create models that could be read by the modeling tools used by Tier1s in the physical ECU design (3) and BSW configuration phases (6), AUTOSAR defines a meta-model that specifies the language for these exchanged models [15]. Therefore, models (A), (B), (D), (F) and (G) represent instances of the AUTOSAR meta-model that specifies their abstract syntax in the UML language. The models itself are serialized into XML (referred to as ARXML, AUTOSAR XML), that represents their concrete syntax, and are validated by the AUTOSAR XML schema that is generated from the AUTOSAR meta-model [16].

In this section, we first describe the AUTOSAR meta-modeling environment in subsection 2.4.1. We then show an example use of the AUTOSAR meta-model in the logical system design (1), physical system design (2), physical ECU design (3) and physical COM design (5) phases in subsection 2.4.2 using our minimalistic system presented in the previous section and show examples of these models in the ARXML syntax. Finally, we show the description of the semantics of the AUTOSAR models described in the AUTOSAR template specifications in subsection 2.4.3.

2.4.1 AUTOSAR meta-modeling environment

As opposed to the commonly accepted meta-modeling hierarchy of MOF [14] that defines 4 layers [5], AUTOSAR modeling environment is described as a five layer hierarchy, as presented below (the names of the layers are taken from the AUTOSAR *Generic Structure* specification [3]):

1. The **ARM4**: MOF 2.0, e.g., the MOF Class
2. The **ARM3**: UML and AUTOSAR UML profile, e.g., the UML Class
3. The **ARM2**: Meta-model, e.g., the SoftwareComponent
4. The **ARM1**: Models, e.g., the WindShieldWiper
5. The **ARM0**: Objects, e.g., the WindShieldWiper in the ECU memory

The mismatch between the number of layers defined by MOF and AUTOSAR lies in the fact that MOF considers only layers connected by the linguistic instantiation (e.g., *SystemSignal* is an instance of UML *Class*), while AUTOSAR considers both linguistic and ontological layers (e.g., *VehicleSpeed* is an instance of *SystemSignal*) [10]. To link these two interpretations of the meta-modeling hierarchy, we can visualize the AUTOSAR meta-modeling hierarchy using two-dimensional representation (known as OCA - Orthogonal Classification Architecture [1]), as shown in Figure 2.8. Linguistic instantiation (L" layers corresponding to MOF layers) are represented vertically and ontological layers ("O" layers) horizontally.

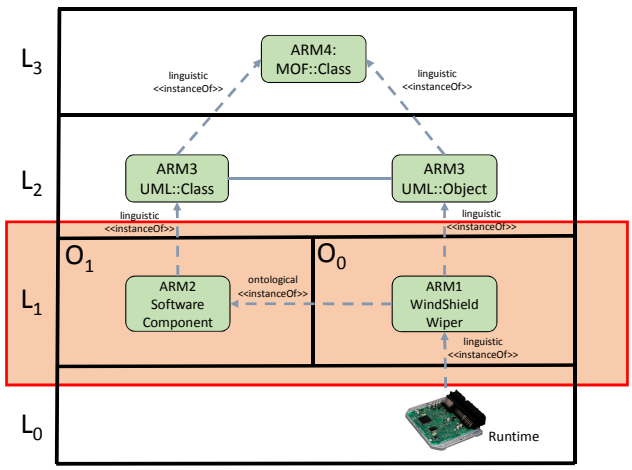


Figure 2.8: AUTOSAR meta-model layers [7]

ARM2 layer is commonly referred to as the "AUTOSAR meta-model" and it ontologically defines, using UML syntax (i.e.,, AUTOSAR meta-model is defined as an instance of UML), AUTOSAR models residing on the *M1* layer (both the AUTOSAR meta-model and AUTOSAR models are located on the *L1* layer). The AUTOSAR meta-model also uses a UML profile that extends the UML meta-model on the *ARM3* layer, which specifies the used stereotypes and tagged values.

Structurally, AUTOSAR meta-model is divided into a number of top-level packages referred to as "templates", where each template defines how to model one part of the system. Modeling semantics, referred to as design requirements and constraints, are described in the template specifications [8].

Probably the most important templates for the design of automotive software systems are the *SWComponentTemplate*, that defines how to model software components and their interaction, *SystemTemplate*, that defines how to model ECUs and their communication, and *ECUCParameterDefTemplate* and *ECUCDescriptionTemplate* that define how to configure ECU basic software. In addition to these templates, AUTOSAR *GenericStructure* template is used to define general concepts (meta-classes) used by all other templates, e.g., handling different variations in the architectural models related to different vehicles. In the next subsection, we provide examples of these templates and AUTOSAR models that instantiate them.

2.4.2 Design based on the AUTOSAR meta-model

A simplified excerpt from *SWComponentTemplate* that is needed for the logical system and physical ECU design of our minimalistic example that calculates the vehicle speed and presents its value to the driver is presented in Figure 2.9.

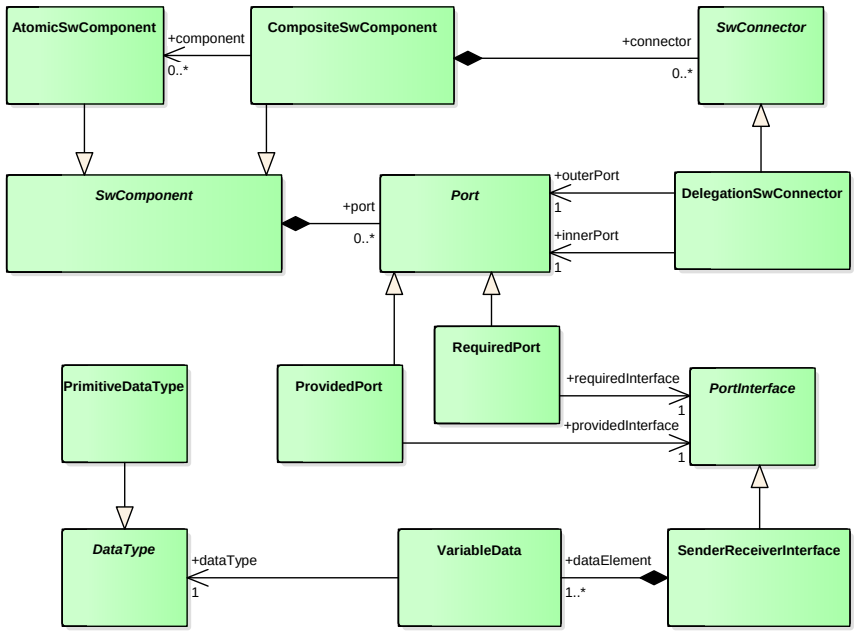


Figure 2.9: Logical and ECU design example (*SwComponentTemplate*)

The excerpt shows the abstract meta-class *SwComponent* that can be either *AtomicSwComponent*, or *CompositeSwComponent* that may refer to multiple *AtomicSwComponent*s. Both types of *SwComponent*s may contain a number of *Port*s that can either be *ProvidedPort*s providing data to the other components in the system, or *RequiredPort*s requiring data from the other components in the system. Ports on the *CompositeSwComponent*s are connected to the ports of the *AtomicSwComponent*s using *DelegationSwConnector*s that belong to the *CompositeSwComponent*s, i.e., *DelegationSwConnector* points to an *outerPort* of the *CompositeSwComponent* and an *innerPort* of the *AtomicSwComponent*. Finally, *Port*s refer to a corresponding *PortInterface*, e.g., *SenderReceiverInterface* or *ClientServerInterface* that contains the actual definition of the *DataType* that is provided or required by this port (e.g., unsigned integer of 32 bits or a struct that consists of an integer and a float).

The model of our example of the logical system design presented in Figure 2.3 that instantiates *SWComponentTemplate* part of the meta-model is shown in Figure 2.10 in the ARXML syntax. We chose ARXML as it is used as exchange format between OEMs and Tier1s, but UML could also be used.

The example shows the definition of the *SpeedCalc* composite software component (lines 1-11) with the *VehicleSpeed* provided port (lines 4-9), and the *Odometer* composite software component (lines 12-22) with the *VehicleSpeed*

```
1    <COMPOSITE-SW-COMPONENT UUID="...">
2       <SHORT-NAME>SpeedCalc</SHORT-NAME>
3       <PORTS>
4           <PROVIDED-PORT UUID="...">
5               <SHORT-NAME>VehicleSpeed</SHORT-NAME>
6               <PROVIDED-INTERFACE-REF DEST="SENDER-RECEIVER-INTERFACE">
7                   /.../VehicleSpeedInterface
8               </PROVIDED-INTERFACE-REF>
9           </PROVIDED-PORT>
10      </PORTS>
11  </COMPOSITE-SW-COMPONENT>
12   <COMPOSITE-SW-COMPONENT UUID="...">
13      <SHORT-NAME>Odometer</SHORT-NAME>
14      <PORTS>
15          <REQUIRED-PORT UUID="...">
16              <SHORT-NAME>VehicleSpeed</SHORT-NAME>
17              <REQUIRED-INTERFACE-REF DEST="SENDER-RECEIVER-INTERFACE">
18                  /.../VehicleSpeedInterface
19              </REQUIRED-INTERFACE-REF>
20          </REQUIRED-PORT>
21      </PORTS>
22  </COMPOSITE-SW-COMPONENT>
23  <SENDER-RECEIVER-INTERFACE UUID="...">
24      <SHORT-NAME>VehicleSpeedInterface</SHORT-NAME>
25      <DATA-ELEMENTS>
26          <VARIABLE-DATA UUID="...">
27              <SHORT-NAME>VehicleSpeed</SHORT-NAME>
28              <DATA-TYPEREF DEST="PRIMITIVE-DATA-TYPE">
29                  /.../UInt16
30              </DATA-TYPE-REF>
31          </VARIABLE-DATA>
32      </DATA-ELEMENTS>
33  </SENDER-RECEIVER-INTERFACE>
34  <PRIMITIVE-DATA-TYPE UUID="...">
35      <SHORT-NAME>UInt16</SHORT-NAME>
36  </PRIMITIVE-DATA-TYPE>
```

Figure 2.10: AUTOSAR model example: logical design

required port (15-20). Both ports refer to the same sender-receiver interface (lines 23-33) that in turn refers to the unsigned integer type of 16 bits (lines 34-36) for the provided/required data.

According to the AUTOSAR methodology, these composite software components are, after their allocation to the chosen ECUs, broken down into a number of atomic software components during the physical ECU design phase. The partial model of our minimalistic example of the physical ECU design presented in Figure 2.5 that instantiates *SWComponentTemplate* part of the meta-model is shown in Figure 2.11 in the ARXML syntax.

The example shows the definition of the *BreakControl* atomic software component (lines 31-41) with the *VehicleSpeed* provided port (lines 34-39) that is referred (lines 12-17) from the *SpeedCalc* composite software component (lines 1-30). We can also see the delegation connector *Delegation1* inside the *SpeedCalc* composite software component (lines 20-28) that connects the provided ports in the *SpeedCalc* and *BreakControl* software components.

A simplified excerpt from *SystemTemplate* that is needed for the physical and COM system design of our minimalistic example is presented in Figure 2.12.

Related to the physical system design, the excerpt shows the *EcuInstance*

```
<COMPOSITE-SW-COMPONENT UUID="...">
    <SHORT-NAME>SpeedCalc</SHORT-NAME>
    <PORTS>
        <PROVIDED-PORT UUID="...">
            <SHORT-NAME>VehicleSpeed</SHORT-NAME>
            <PROVIDED-INTERFACE-REF DEST="SENDER-RECEIVER-INTERFACE">
                /.../VehicleSpeedInterface
            </PROVIDED-INTERFACE-REF>
        </PROVIDED-PORT>
    </PORTS>
    <COMPONENTS>
        <COMPONENT>
            <SHORT-NAME>BreakControl</SHORT-NAME>
            <COMPONENT-REF DEST="ATOMIC-SW-COMPONENT">
                /.../BreakControl
            </COMPONENT-REF>
        </COMPONENT>
    </COMPONENTS>
    <CONNECTORS>
        <DELEGATION-SW-CONNECTOR UUID="...">
            <SHORT-NAME>Delegation1</SHORT-NAME>
            <INNER-PORT-REF DEST="P-PORT-PROTOTYPE">
                /.../BreakControl/VehicleSpeed
            </INNER-PORT-REF>
            <OUTER-PORT-REF DEST="P-PORT-PROTOTYPE">
                /.../SpeedCalc/VehicleSpeed
            </OUTER-PORT-REF>
        </DELEGATION-SW-CONNECTOR>
    </CONNECTORS>
</COMPOSITE-SW-COMPONENT>
<ATOMIC-SW-COMPONENT UUID="...">
    <SHORT-NAME>BreakControl</SHORT-NAME>
    <PORTS>
        <PROVIDED-PORT UUID="...">
            <SHORT-NAME>VehicleSpeed</SHORT-NAME>
            <PROVIDED-INTERFACE-REF DEST="SENDER-RECEIVER-INTERFACE">
                /.../VehicleSpeedInterface
            </PROVIDED-INTERFACE-REF>
        </PROVIDED-PORT>
    </PORTS>
</ATOMIC-SW-COMPONENT>
```

Figure 2.11: AUTOSAR model example: ECU design

meta-class with *diagnosticAddress* attribute that may contain a number of *CommunicationConnector*s that represent connections of the *EcuIstance* to a *PhysicalChannel* (e.g., *CanCommunicationConnector* connects one *EcuInstance* to a *CanPhysicalChannel*). A number of *SwComponent*s (*CompositeSwComponent*s or *AtomicSwComponent*s) created in the logical design can be allocated onto one *EcuInstace* by means of *SwcToEcuMapping*s.

Related to the physical COM design, the excerpt shows the *SenderReceiverToSignalMapping* of the *VariableData* created in the logical design to a *SystemSignal*. It also shows that one *SystemSignal* can be sent to multiple buses by means of creating different *ISignal*s and mapping them to *IPdu*s that are in turn mapped to *Frame*s. *IPdu* is one type of a *Pdu* (Protocol Data Unit) that is used for transporting signals and there may be other types of *Pdu*s, e.g., *DcmPdu* for transporting diagnostic messages.

The model of our example of the physical system design presented in Figure 2.4 that instantiates *SystemTemplate* part of the meta-model is shown in

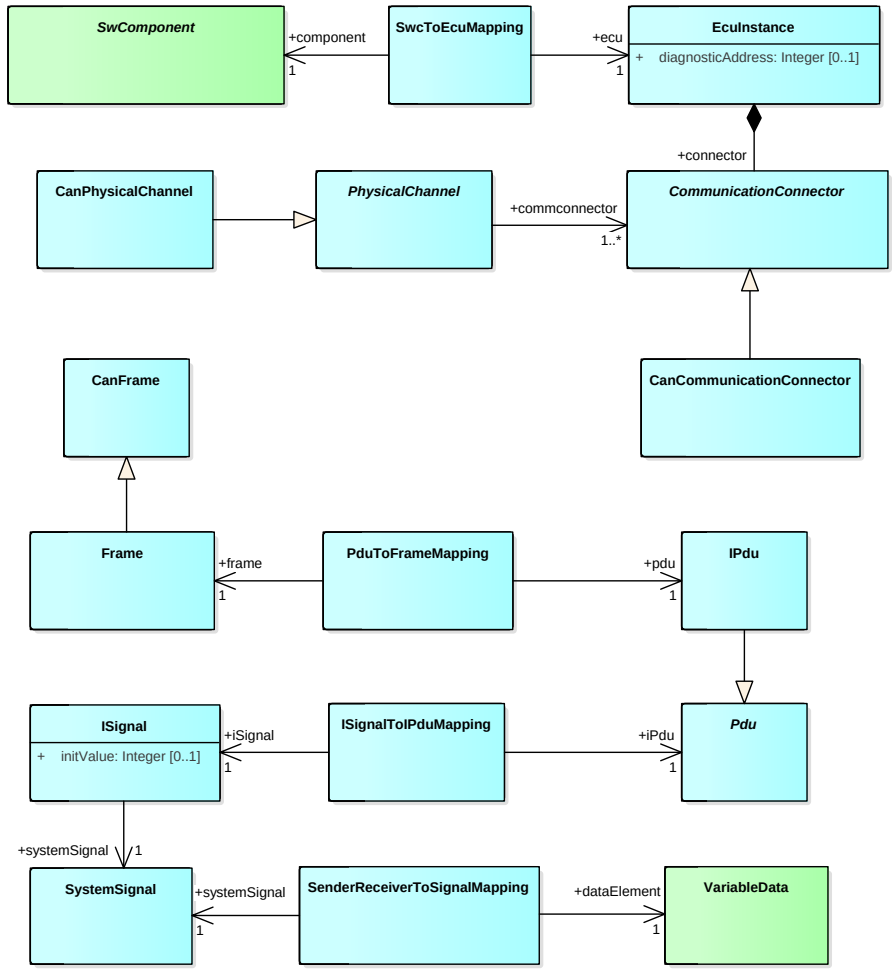


Figure 2.12: Physical and COM design (*SystemTemplate*)

Figure 2.13.

The example shows the definition of the *BreakControl* ECU with diagnostic address 10 (lines 1-9) that owns a CAN communication connector (lines 5-7). It also shows the mapping of the *SpeedCalc* composite software component onto *BreakControl* ECU (lines 10-14). Finally, it shows the definition of the *Can1* physical channel (lines 15-24) that points to the CAN communication connector of the *BreakControl* ECU (lines 19-21), thereby indicating that this ECU is connected to *Can1*.

The model of our example of the COM system design presented in Figure 2.6 that instantiates *SystemTemplate* part of the meta-model is shown in Figure 2.14.

The example shows the definition of the *VehicleSpeed* system signal (lines 1-3) that is mapped to the *SpeedCalc* variable data element defined in the logical design phase (lines 4-12). The example also shows the creation of the *ISignal VehicleSpeedCan1* (lines 13-19) with initial value of 0 that is meant to transmit

```
<ECU-INSTANCE UUID="...">
    <SHORT-NAME>BreakControl</SHORT-NAME>
    <ECU-ADDRESS>10</ECU-ADDRESS>
    <CONNECTORS>
        <CAN-COMMUNICATION-CONNECTOR UUID="...">
            <SHORT-NAME>Can1Connector</SHORT-NAME>
        </CAN-COMMUNICATION-CONNECTOR>
    </CONNECTORS>
</ECU-INSTANCE>
<SWC-TO-ECU-MAPPING UUID="...">
    <SHORT-NAME>Mapping1</SHORT-NAME>
    <COMPONENT-REF DEST="SW-COMPONENT">/.../SpeedCalc</SW-REF>
    <ECU-REF DEST="ECU-INSTANCE">/.../BreakControl</ECU-REF>
</SWC-TO-ECU-MAPPING>
<CAN-PHYSICAL-CHANNEL UUID="...">
    <SHORT-NAME>Can1</SHORT-NAME>
    <COMM-CONNECTORS>
        <COMMUNICATION-CONNECTOR-REF-CONDITIONAL>
            <COMMUNICATION-CONNECTOR-REF DEST="CAN-COMMUNICATION-CONNECTOR">
                /.../BreakControl/Can1Connector
            </COMMUNICATION-CONNECTOR-REF>
        </COMMUNICATION-CONNECTOR-REF-CONDITIONAL>
    </COMM-CONNECTORS>
</CAN-PHYSICAL-CHANNEL>
```

Figure 2.13: AUTOSAR model example: Physical design

```
<SYSTEM-SYGNAL UUID="...">
    <SHORT-NAME>VehicleSpeed</SHORT-NAME>
</SYSTEM-SYGNAL>
<SENDER-RECEIVER-TO-SIGNAL-MAPPING UUID="...">
    <SHORT-NAME>Mapping2</SHORT-NAME>
    <DATA-ELEMENT-REF DEST="VARIABLE-DATA">
        /.../VehicleSpeedInterface/SpeedCalc
    </DATA-ELEMENT-REF>
    <SYSTEM-SIGNAL-REF DEST="SYSTEM-SIGNAL">
        /.../VehicleSpeed
    </SYSTEM-SIGNAL-REF>
</SENDER-RECEIVER-TO-SIGNAL-MAPPING>
<I-SYGNAL UUID="...">
    <SHORT-NAME>VehicleSpeedCan1</SHORT-NAME>
    <INIT-VALUE>0</INIT-VALUE>
    <SYSTEM-SIGNAL-REF DEST="SYSTEM-SIGNAL">
        /.../VehicleSpeed
    </SYSTEM-SIGNAL-REF>
</I-SYGNAL>
<I-PDU UUID="...">
    <SHORT-NAME>IPdu1</SHORT-NAME>
</I-PDU>
<I-SIGNAL-TO-I-PDU-MAPPING UUID="...">
    <SHORT-NAME>Mapping3</SHORT-NAME>
    <I-PDU-REF DEST="I-PDU">/.../IPdu1</I-PDU-REF>
    <I-SIGNAL-REF DEST="I-SIGNAL">/.../VehicleSpeedCan1</I-SIGNAL-REF>
</I-SIGNAL-TO-I-PDU-MAPPING>
<CAN-FRAME UUID="...">
    <SHORT-NAME>CanFrame1</SHORT-NAME>
</CAN-FRAME>
<I-PDU-TO-FRAME-MAPPING UUID="...">
    <SHORT-NAME>Mapping4</SHORT-NAME>
    <PDU-REF DEST="I-PDU">/.../IPdu1</PDU-REF>
    <FRAME-REF DEST="CAN-FRAME">/.../CanFrame1</FRAME-REF>
</I-PDU-TO-FRAME-MAPPING>
```

Figure 2.14: AUTOSAR model example: COM design

the vehicle speed on the *Can1* bus defined in the physical design phase. This *ISignal* is mapped to *Pdu1* (lines 20-22) using *ISignalToIPduMapping* (23-27) that in turn is mapped to *CanFrame1* (lines 28-30) using *IPduToFrameMapping* (lines 31-35).

2.4.3 AUTOSAR template specifications

As other language definitions, the AUTOSAR meta-model defines only syntax for different types of architectural models without explaining how its meta-classes shall be used to achieve certain semantics. This is done in the natural language specifications called templates [8], e.g., *SwComponentTemplate* and *SystemTemplate*, that are explaining different parts of the AUTOSAR meta-model. These templates consist of the following main items:

- Design requirements to be fulfilled by the models (specification items).
- Constraints to be fulfilled by the models and checked by modeling tools.
- Figures explaining the use of a group of meta-classes.
- Class tables explaining meta-classes and their attributes/connectors.

As an example of a specification item related to our minimalistic example that calculates vehicle speed and presents its value to the driver, we present specification item no. 01009 from the *SystemTemplate* that describes the use of *CommunicationConnector*s:

[TPS_SYST_01009] Definition of CommunicationConnector [An EcuInstance uses CommunicationConnector elements in order to describe its bus interfaces and to specify the sending/receiving behavior.]

As an example of a constraint, we present constraint no. 1032 from the *SwComponentTemplate* that describes the limitation in the use of *DelegationSwConnector*s.

[constr_1032] DelegationSwConnector can only connect Ports of the same kind [A DelegationSwConnector can only connect Ports of the same kind, i.e. ProvidedPort to ProvidedPort and RequiredPort to RequiredPort.]

The majority of constraints including *constr_1032* could be specified directly in the AUTOSAR meta-model using OCL (Object Constraint Language). However due to the complexity of OCL and thousands of automotive engineers in more than hundred OEM and supplier companies that develop automotive software components based on AUTOSAR, natural language specifications are considered a better approach for such a wide audience [15].

Meta-model figures show relationships between a number of meta-classes using UML notation and they are similar to figures 2.9 and 2.12 presented in the previous section. These figures are usually followed by class tables that describe the meta-classes in the figures in more details, e.g. description of the meta-classes, their parent classes and attributes/connectors, so that the

readers of the AUTOSAR specification do not need to look directly into the AUTOSAR meta-model which is maintained in the *Enterprise Architect* tool.

In addition to specification items, constraints, figures and class tables, AUTOSAR template specifications also contain a substantial amount of plain text that provides additional explanations, e.g., introductions to the topic and notes after specification items and constraints.

2.5 AUTOSAR ECU middleware

AUTOSAR provides detailed functional specifications for the modules of its middleware layer (basic software modules). For example, *COM* specification describes the functionality of the Communication Manager module that is mostly responsible for handling the communication between ECUs, i.e., transmitting signals received from the RTE onto electronic buses and vice-versa. These specifications consist of the following main items:

- Functional requirements that should be fulfilled by the implementation of the BSW modules.
- Description of APIs of the BSW modules.
- Sequence diagrams explaining the interaction between BSW modules.
- Configuration parameters used for configuring the BSW modules.

Functional side of the AUTOSAR BSW module specifications (functional requirements, APIs and sequence diagrams) is outside of the scope of this chapter. However, we do describe here the general approach to configuration of the BSW modules as it is done based on the AUTOSAR meta-model and its templates.

Two of the AUTOSAR templates are responsible for specifying configuration of the AUTOSAR basic software - *ECUCParameterDefTemplate* and *ECUCDescriptionTemplate* on the *ARM2* layer. *ECUCParameterDefTemplate* specifies the general definition of configuration parameters, e.g., that parameters can be grouped into containers of parameters and that they can be configured at different configuration times (e.g., before or after building the complete ECU software). *ECUCDescriptionTemplate* specifies modeling of concrete parameter and container values that reference their corresponding definitions from the *ECUCParameterDefTemplate*.

The values of configuration parameters from the *ECUCDescriptionTemplate* models can be automatically derived from the models of other templates, e.g., *SoftwareComponentTemplate* and *SystemTemplate*. This process is called "upstream mapping" and it can be done automatically with the support from the ECU configuration tools [11]. A simplified example of the *ECUCParameterDefTemplate* and *ECUCParameterDefTemplate* and their models including the upstream mapping process is shown in Figure 2.15 in UML syntax.

The *ECUCParameterDefTemplate* on the *ARM2* layer (left blue box) specifies modeling of the definition of configuration parameters (*ECUCParameterDefs*) and containers (*ECUCContainerDefs*), with an example of the integer parameter definition (*ECUCIntegerParameterDef*). The *ECUCDescriptionTemplate* (left yellow box) specifies modeling of the values of containers

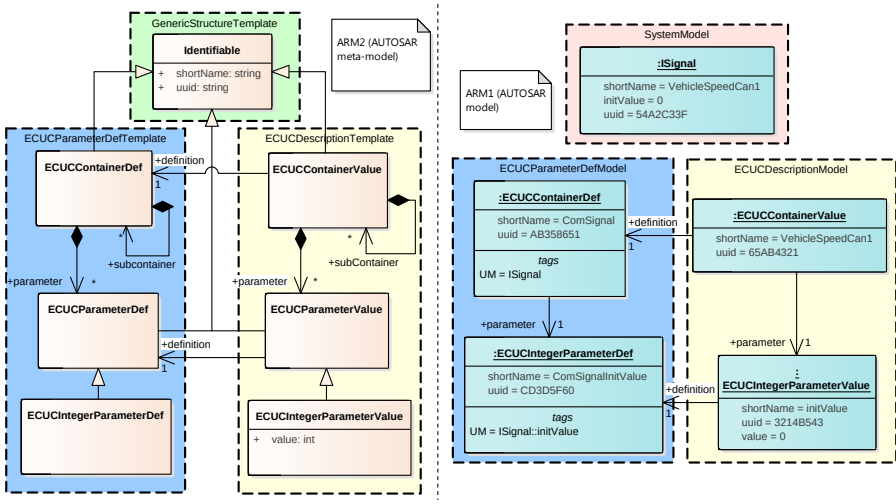


Figure 2.15: Example of the AUTOSAR templates and their models

(*ECUCContainerValue*s) and parameters (*ECUCParameterValue*s), with an example of the integer parameter value (*ECUCIntegerParameterValue*). As with the elements from the *SwComponentTemplate* and *SystemTemplate*, the elements from these two templates are also inherited from the common element in the *GenericStructureTemplate* (green box) named *Identifiable*, which provides them with the short name and unique identifier (UUID).

The standardized model (i.e., provided by AUTOSAR) of the *ECUCParameterDefTemplate* can be seen on the *ARM1* layer (right blue box). It shows the *ECUCContainerDef* instance with *shortName* "ComSignal" that refers to the *ECUCParameterDef* instance with *shortName* "ComSignalInitValue". These two elements both have the tagged value named *UM*, denoting Upstream Mapping. The *UM* tagged value for the "ComSignal" container instance refers to the *ISignal* meta-class from the *SystemTemplate*. The *UM* tagged value for the "ComSignalInitValue" parameter instance refers to the *initValue* attribute of the *ISignal*. This implies that for every *ISignal* instance in the *SystemModel*, one *ECUCContainerValue* instance in the *ECUCDescriptionModel* shall be created with an *ECUCParameterValue* instance. The value of this parameter instance shall be equal to the *initValue* attribute of that *SystemSignal* instance.

Considering the "VehicleSpeedCan1" *ISignal* with "initValue" 0 (orange box) that we defined in our *SystemModel* shown in Figure 2.14 (COM design phase), the *ECUCDescriptionModel* (right yellow box) can be generated. This model contains one instance of the *ECUCContainerValue* with *shortName* "VehicleSpeedCan1" that is defined by the "ComSignal" container definition and refers to one instance of the *ECUCParameterValue* with *shortName* "initValue" of value 0 that is defined by the "ComSignalInitValue" parameter definition.

AUTOSAR provides the standardized *ARM1* models of the *ECUCParameterDefTemplate* for all configuration parameters and containers of the ECU basic software. For example, *ComSignal* container with *ComSignalInitValue*

are standardized for the *COM* BSW module. On the smallest granularity, standardized models of the *ECUCParameterDefTemplate* are divided into a number of packages, where each package contains configuration parameters of one *Basic software* module. On the highest granularity, these models are divided into different logical packages, including ECU communication, diagnostics, memory access and IO access.

Bibliography

[1] C. Atkinson and T. Kühne. "Model-Driven Development: A Metamodeling Foundation". In: *Journal of IEEE Software* 20.5 (2003), pp. 36–41.

[2] *Automotive Open System Architecture*. AUTOSAR. 2003. URL: www.autosar.org.

[3] *AUTOSAR Generic Structure Template v4.2.1*. 2014. URL: www.autosar.org.

[4] *AUTOSAR Layered Software Architecture v4.2.1*. 2013. URL: www.autosar.org.

[5] Jean Bézivin and Olivier Gerbé. "Towards a Precise Definition of the OMG/MDA Framework". In: *International Conference on Automated Software Engineering*. 2001, pp. 273–280.

[6] M. Broy et al. "Engineering Automotive Software". In: *Proceedings of the IEEE*. Vol. 95. 2. 2007.

[7] D. Durisic et al. "Addressing the Need for Strict Meta-Modeling in Practice - A Case Study of AUTOSAR". In: *International Conference on Model-Driven Engineering and Software Development*. 2016.

[8] P. Gouriet. "Involving AUTOSAR Rules for Mechatronic System Design". In: *International Conference on Complex Systems Design & Management*. 2010, pp. 305–316.

[9] P. Kruchten. "Architectural Blueprints - The "4+1" View Model of Software Architecture". In: *IEEE Softwar* 12.6 (1995), pp. 42–50.

[10] T. Kühne. "Matters of (Meta-) Modeling". In: *Journal of Software and Systems Modeling* 5.4 (2006), pp. 369–385.

[11] J. C. Lee and T. M. Han. "ECU Configuration Framework Based on AUTOSAR ECU Configuration Metamodel". In: *International Conference on Convergence and Hybrid Information Technology*. 2009, pp. 260–263.

[12] Y. Liu, Y. Q. Li, and R. K. Zhuang. "The Application of Automatic Code Generation Technology in the Development of the Automotive Electronics Software". In: *International Conference on Mechatronics and Industrial Informatics Conference*. Vol. 321-324. 2013, pp. 1574–1577.

[13] *MDA guide 2.0*. Object Management Group. 2014. URL: http://www.omg.org/mda/.

[14] *MOF 2.0 Core Specification*. Object Management Group. 2004. URL: www.omg.org.

[15] G. Nordstrom et al. "Metamodeling - Rapid Design and Evolution of Domain-Specific Modeling Environments". In: *IEEE Conference on Engineering of Computer Based Systems*. 1999, pp. 68–74.

[16] M. Pagel and M. Brörkens. "Definition and Generation of Data Exchange Formats in AUTOSAR". In: *European Conference on Model Driven Architecture, Foundations and Applications*. 2006, pp. 52–65.