KTH Industrial Engineering
and Management

Supporting Model Evolution in Model-Driven Development of Automotive Embedded Systems

MATTHIAS BIEHL

Licentiate Thesis
Stockholm, Sweden, 2010

TRITA-MMK 2010:08
ISSN 1400-1179
ISRN/KTH/MMK/R-10/08-SE
ISBN 978-91-7415-723-9

KTH School of Industrial
Engineering and Management
10044 Stockholm
Sweden

Academic thesis, which with the approval of the Royal Institute of Technology, will be presented for public review in fulfillment of the requirements for a Licentiate of Technology in Machine Design. The public review is held in Room A325, Department for Machine Design, Royal Institute of Technology, Brinellvägen 83, Stockholm on 2010-11-25 9:15.

Print: US-AB

Abstract

Innovative functions in cars, such as active safety systems and advanced driver assistance systems, are realized as embedded systems. The development of such automotive embedded systems is challenging in several respects: the product typically has several crosscutting system properties, experts of diverse disciplines need to cooperate and appropriate processes and tools are required to improve the efficiency and the complexity management of development. Model-driven development captures the architecture of the embedded system in the form of models with well-defined metamodels.

Model-driven development provides a partial solution to some of the challenges of embedded systems development, but it also introduces new challenges. Models do not remain static, but they change over time and evolve. Evolution can change models in two ways: (1) by making design decisions and adding, deleting or changing model elements, or (2) by reusing models in different tools. We propose support for both aspects of model evolution.

(1) When models are changed, the design decisions and the justification for the change are usually neither captured nor documented in a systematic way. As a result, important information about the model is lost, making the model more difficult to understand, which hampers model evolution and maintenance. To support model evolution, design decisions need to be captured explicitly using an appropriate representation. This representation reduces the overhead of capturing design decisions, keeps the model and the design decision documentation consistent and links the design decision documentation to the model. As a result, the captured design decisions provide a record of the model evolution and the rationale of the evolution.

(2) Several models and views are used to describe an embedded system in different life cycle stages and from the viewpoints of the involved disciplines. To create the various models, a number of specialized development tools are used. These tools are usually disconnected, so the models cannot be transferred between different tools. Thus, models may become inconsistent, which hampers understandability of the models and increases the cost of development. We present a model-based tool integration approach that uses a common metamodel in combination with model transformation technology to build bridges between different development tools. We apply this approach in a case study and integrate several tools for automotive embedded systems development: A systems engineering tool, a safety engineering tool and a simulation tool.

As a part of future work, we plan to extend the tool integration approach to exchange not only models but also the attached documentation of design decisions. As a result, the design decision documentation is linked consistently to corresponding model elements of the various tool-specific models, supporting model evolution across several development tools.

Acknowledgements

Many people have been involved in the work behind the thesis. I would like to thank:

- Martin Törngren for the opportunity to work towards the licentiate thesis and for providing advise, feedback, encouragement and enthusiasm throughout the work!
- De-Jiu Chen and Carl-Johan Sjöstedt for the teamwork in the ATESST2 project and for the contributions and feedback on the co-authored publications.
- Jad El-khoury for providing detailed feedback on the papers.
- Tahir Naseer Qureshi, Magnus Persson, Lei Feng, Sagar Behere, Fredrik Asplund, Alex Schenkman, Begashaw Gezu Kirsie and Kristian Gustafsson for the great working atmosphere and discussions.
- All other colleagues in the department of machine design and in the research projects ATESST2, CESAR and iFEST.
- My parents, sisters and friends for their support.

Matthias Biehl
Stockholm, November 2010

Contents

List of Appended Publications

This thesis is based on six papers that are appended to this thesis. An overview of the contents of these papers is presented in chapter 4.

- Matthias Biehl, "Literature Study on Model Transformations," Technical Report, Royal Institute of Technology, ISSN 1400-1179, ISRN/KTH/MMK/R-10/07-SE, Stockholm, Sweden, July 2010. Included as Paper A.

- Matthias Biehl, Chen De-Jiu, and Martin Törngren, "Integrating Safety Analysis into the Model-based Development Toolchain of Automotive Embedded Systems," in *Proceedings of the ACM SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems (LCTES 2010)*, April 2010, pp. 125+. Included as Paper B.

  Chen developed the EAST-ADL2 metamodel for error modeling, Matthias developed the concepts and implementation for tool integration, performed the case-study and wrote the paper, Chen and Martin provided essential feedback.

- Matthias Biehl, Carl-Johan Sjöstedt, and Martin Törngren, "A Modular Tool Integration Approach - Experiences from two Case Studies," in *3rd Workshop on Model-Driven Tool & Process Integration (MDTPI2010)*, June 2010. Included as Paper C.

  Carl-Johan performed the case study on integrating Simulink and wrote section 4.2, Matthias performed the case study on integrating the safety analysis tool HiP-HOPS and wrote the rest of the paper, Martin provided essential feedback.

- Matthias Biehl, "Literature Study on Design Rationale and Design Decision Documentation for Architecture Descriptions," Technical Report, ISSN 1400-1179, ISRN/KTH/MMK/R-10/06-SE, Royal Institute of Technology, Stockholm, Sweden, July 2010. Included as Paper D.

- Matthias Biehl and Martin Törngren, "An Executable Design Decision Representation using Model Transformations," in *36th EUROMICRO Conference*

on Software Engineering and Advanced Applications (SEAA2010), September 2010. Included as paper E.

Matthias developed the concepts and prototypes, performed the case-study and wrote the paper, Martin provided essential feedback.

- Matthias Biehl, "Documenting Stepwise Model Refinement using Executable Design Decisions," in *International Workshop on Models and Evolution (ME 2010)*, October 2010. Included as paper F.

List of Other Publications

- Matthias Biehl and Welf Löwe, "Automated Architecture Consistency Checking for Model Driven Software Development," in *Proceedings of the Fifth International Conference on the Quality of Software Architectures (QoSA 2009)*, June 2009, pp. 36-51. [Online]. Available: `http://dx.doi.org/10.1007/978-3-642-02351-4_3`.

- Ulf Sellgren, Martin Törngren, Diana Malvius, and Matthias Biehl, "PLM for Mechatronics Integration," in *Proceedings of the 6th International Product Lifecycle Management Conference (PLM 2009)*, July 2009. [Online]. Available: `http://www.md.kth.se/~ulfs/Publications/PLMForMechatronics.pdf`.

- Olaf Seng, Markus Bauer, Matthias Biehl, and Gert Pache, "Search-based Improvement of Subsystem Decompositions," in *Proceedings of the 2005 ACM Conference on Genetic and Evolutionary Computation (GECCO 2005)*, 2005, pp. 1045-1051. [Online]. Available: `http://dx.doi.org/10.1145/1068009.1068186`.

- ATESST2 Project Deliverables D2.1, D3.1, D3.2, D3.5, D4.3.1 [Online]. Available: `http://www.atesst.org`.

- iFEST Project Deliverable D2.1 [Online].
  Available: `http://www.artemis-ifest.eu`.

Chapter 1

Introduction

1.1 Automotive Embedded Systems Development

Innovative functions in cars, such as active safety systems and advanced driver assistance systems, are realized as automotive embedded systems. On the one hand, automotive embedded systems have a great potential. On the other hand, the development of automotive embedded systems is complex. This thesis focuses on methods and tools that aim to support dealing with this complexity.

A number of factors contribute to the complexity of development: the product, the people and their organization and the processes and tools [75]. The embedded system as a product is complex, as it needs to be integrated into a larger system and fulfill a large number of non-functional cross-cutting system requirements [144]. Embedded systems development is multidisciplinary [30], as people with expertise in several disciplines are required. Processes, methods and tools structure the development work. We will examine how these factors influence the complexity of development.

1.1.1 Product: Embedded Systems

The IEEE Glossary defines an *embedded computer system* as a "computer system that is part of a larger system and performs some of the requirements of that system" [109]. Embedded systems are often classified according to the domains of these larger systems they are part of, for example automotive, automation, aerospace or rail. In this thesis we focus on automotive embedded systems.

According to the IEEE definition, an embedded system is tightly integrated into a larger system. The integration results typically in many non-functional requirements for embedded systems that go beyond those of IT systems [144, 69]:

- Real-time constraints: embedded systems must deliver results within a maximum time under all circumstances.

- Reliability: embedded systems run for a long time without service and unexpected behavior of the embedded system might damage the environment.
- Safety criticality: embedded systems impact people and constitute potential safety hazards.
- Security: embedded systems are increasingly interconnected with each other, but also with sensors and actuators. Security protection is a major issue.
- Limited resources: embedded systems often need to be low cost devices, which entails limited resources. This imposes constraints on the computation power, available memory and consumable electrical power.
- Long lifetime: embedded systems are built into mechanical products that have a long expected lifetime.
- Heterogeneity: embedded systems need to support different types of hardware, so they need to be flexible, adaptable and portable to new hardware.

Requirements, such as security and safety, cannot be realized as system properties in an isolated way, but the interactions among these system properties need to be considered [68]. Out of several such relatively simple interactions, complex patterns arise. This phenomenon is called *emergence* [210], where new structures or behaviors of the system arise that cannot be reduced to those of the isolated parts. These factors contribute to the complexity of developing embedded systems.

1.1.2 People and Organization: Multidisciplinarity

The development of embedded systems is multidisciplinary[1], as the embedded system is influenced by the surrounding system and also itself influences the surrounding system [68]. The surrounding system might be a mechanical, electrical or electronic system. People of different educational backgrounds i.e. control engineers, mechanical engineers, electrical engineers and software engineers work together on creating one product that must meet the needs of the users [30]. Often the people of one discipline do not understand the specialties of the other disciplines any more [164]. Each of these disciplines has its separate and dedicated set of tools, models and views that support development, communication, analysis and simulation [39].

The different aspects of the system, its mechanical, electrical and software parts need to fit together, when they are integrated into one product. The foundations for this integration are already laid in the early stages of system development, when the architecture of the system is specified. The complexity of this integration is further aggravated by a number of non-functional requirements and important crosscutting system properties such as safety. A crosscutting system property cannot be resolved

[1] In *interdisciplinary* areas the disciplines are integrated, while in *multidisciplinary* areas the cooperation is not interactive and disciplines remain separate [7].

for each system aspect such as mechanics, software or electronics separately, but interactions between these different disciplines need to be considered and resolved.

Despite the differences the engineers need to communicate their design and architecture effectively across the disciplines. A *boundary object* [193] is a description of information that is a common reference for different disciplines. Boundary objects have a bridging function and serve as a basis for communication between the members of different disciplines.

1.1.3 Process and Tools

Processes and tools are proposed both by academia and by industry. Academic processes usually provide a reference and clean terminology, but are often difficult to apply in practice. Industrial processes describe the current state of practice in development, but sometimes the terminology is not unambiguously defined.

Many academic process models for general engineering design and development have been proposed. They have their roots in mechanical engineering, systems engineering and mathematics. Examples are Konstruktionslehre by Pahl/Beitz [172], Product Development by Ulrich and Eppinger [211], Axiomatic Design by Suh [195, 196], Function-Behavior-Structure Model by Gero [89], Theory of Technical Systems by Hubka/Eder [106, 107], Concept-Knowledge Theory [97] and VDI 2222 [212]. Specifically relevant for embedded systems is the Y-chart invented Gajski and Kuhn [86, 90]. The model defines five abstraction levels, each of them can be specified by three models describing behavior, structure and geometry.

Industrially applied development processes for embedded systems typically follow the V-Model described in VDI 2206 [213]. In addition, approaches for complexity management [205] are used, such as model-based/model-driven development [164] and component-based development [54]. Model-based and model-driven development is introduced in section 2.1.

Processes, methods and tools are closely related and dependent [164]. Processes can be decomposed into tasks. Methods describe how the tasks of a process are implemented.

Methods can be supported by development tools, for example to improve the efficiency of a method. A list of commonly used development tools for embedded software development is provided in [68]. In the context of this work we use the term *tool* synonymously to a *development tool*, which is a software program that supports the development process.

1.1.4 Summary

The complexity of the development of embedded systems is determined by the complexity of the factors product, organization, processes and tools. We can distinguish between the essential and accidental complexity of development [33]. *Essential complexity* is the complexity that lies in the nature of the problem and the complexity cannot be influenced without changing the problem. *Accidental complexity* is the

complexity that is introduced due to imperfect choices in any of the factors and it can be improved upon.

In this thesis we assume that a model-driven development approach for embedded systems is applied and we study and develop tools that are intended to reduce the accidental complexity. We focus on tools for automotive embedded software development as a means for increasing the efficiency and reducing the cost of development.

1.2 Problem Formulation

Model-driven development is an approach for managing the complexity of development. Some, but not all of the issues for managing the complexity of embedded systems development identified in the previous section are addressed by a model-driven development approach.

- *Product:* Model-driven development creates models, which allows analyzing and simulating the embedded system early in the development process, when corrective *changes* are relatively straightforward and cheap.

- *People:* The multidisciplinary aspects of embedded systems development is reflected in model-driven development by multiple views and models of the same system. However independent *changes* of these models during development and maintenance often lead to inconsistencies between the models.

- *Process and Tools:* There is no established industrial development process for model-driven development. Also, the available development tools are often immature: The tools do not interoperate well; it is often not possible to *change* the development tool and load an existing model in another development tool. The development tools do not capture and document all relevant metadata, such as the reasons and design decisions associated with *changes* in the model.

In summary, model-driven approaches promise a partial solution to some of the challenges of embedded systems development, but it brings up new challenges. One of these challenges is the handling of *changes* in model-driven development: models do not remain static, but they change over time and *evolve*. In a development process, the change can affect the content, nature, structure or representation of models. Here we present the challenge of model evolution in more depth:

- *Evolution of multiple views:* The models of different views are changed independently during development and maintenance tasks, leading to *inconsistencies* between the models. As a result, the system becomes harder to understand, making further maintenance and integration more complicated and costly. Perry and Wolf discuss this problem of views that drift apart as *architectural drift*, which may lead to a deterioration and erosion of the model [175, 206].

- *Evolution of models in multi-tool environments:* Each engineering discipline prefers a different set of COTS (commercial off the shelf) development tools that excel in that particular discipline [73] or life cycle stage. However, these tools do not interoperate well and models created in one tool cannot be readily used in another, similar tool. This is, however, desirable for the evolution of models representing multiple views of the system. The lack of tool interoperability leads to gaps in the development process, manual translations of models and, in the end, to inefficiency and high development costs [185]. Models need to be adapted in such a way that they can be developed using several tools, even if these tools expect the model to conform to different tool-specific metamodels. Reusing a model in another development tool than the model was originally developed in, is just another step in the evolution of the model. The model needs to evolve by adapting its metamodel to the tool-specific metamodel of the new tool.
- *Documentation of design decisions to capture model evolution:* When models are changed, the design decisions and the justification for the change are usually not captured or documented in a systematic way. As a result, important information about the model is lost, making the model more difficult to understand, which hampers maintenance tasks. To support the model evolution, design decisions need to be made explicit, and an appropriate representation for design decisions needs to be found that represents design decisions explicitly, reduces the overhead of capturing design decisions, keeps the model and the design decision documentation consistent and links the design decision to the model.
- *Consistent documentation of design decisions in multi view models:* If models are available in multiple views, not only the models but also the documentation of the models needs to be consistent between the views. If a model is created automatically based on information from another model, a documentation might not be available for the newly created model and might need to be created manually.

1.2.1 Illustrative Example

Let us examine the problem in a simplified development scenario (see figure 1.1). A model of a new automotive brake-by-wire system is developed in MATLAB/Simulink (c.f. figure 1.2). Several design decisions are made while the Simulink model is refined, among them is the decision to replicate a subsystem to increase its reliability. After successful simulation of the model, a team of software developers will develop the corresponding source code. From the structure of the existing Simulink model they may generate source code directly or create a UML model. Tool integration technology translates the Simulink model into a corresponding UML model [23]. When the software engineers receive the UML model (see figure 1.2), they do not find any explanation for the duplicated `PedalSensor` elements, which have been

introduced intentionally into the Simulink model to improve the reliability of the brake system. However, this information is not expressed in the automatically generated UML model. Consequently the software engineers might remove one of the `PedalSensors` from the model, as they perceive it as an unnecessary redundancy.

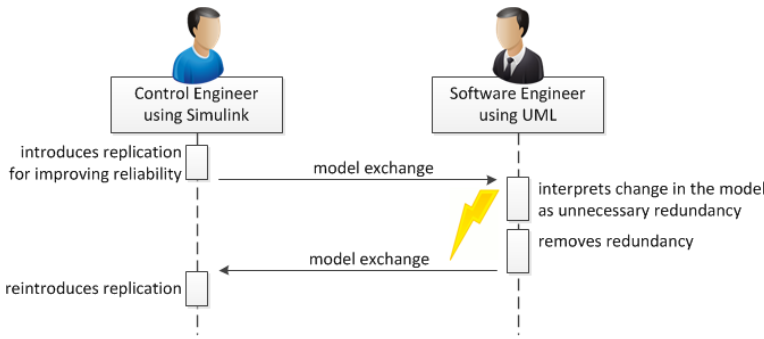

Figure 1.1: Model exchange without design decision documentation (notation used: UML sequence diagram)

The design decisions that were made and documented in the Simulink tool are not available when working with the automatically translated UML model. As a consequence, inconsistencies between models in multi-tool environments might be created, when models are translated automatically.

The example illustrates that there is a need for supporting the integration of development tools, as well as for documenting the design decisions made during development.

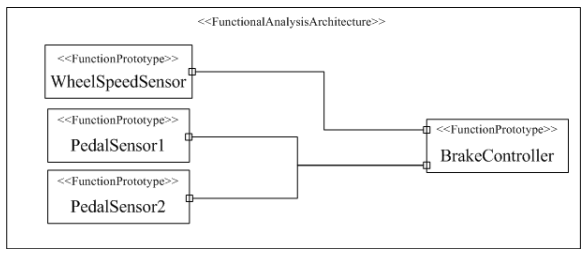

Figure 1.2: Simplified brake-by-wire system with double redundancy

1.3 Objectives and Assumptions

The goal of this work is evaluating and improving the evolution of models in the context of model-driven development of automotive embedded systems. We focus on two aspects of model evolution:

- A model can evolve by changing the metamodel it conforms to. We study this aspect of model evolution in the context of tool integration.
- A model can evolve by adding, deleting or changing model elements. We study this aspect of model evolution in the context of design decision documentation.

We distinguish the following objectives:

1. Models need to be automatically adapted to tool-specific metamodel. This allows using models seamlessly in other tools than the model was originally created in. Both concepts and tools realizing tool-integration targeted for the domain of automotive embedded systems need to be developed.
2. The design decisions leading to changes in models need to be documented and linked to the model in a consistent manner. Both concepts and tools for supporting design decision documentation need to be developed.

Besides the assumption of model-driven development to improve complexity management and development efficiency, we share the following assumptions with the involved research communities. The assumption of the tool integration research community is that the integration of development tools leads to increased productivity of heterogeneous engineering teams and higher quality of the developed products [221]. It is assumed that design decision documentation will help to manage evolution and improve the quality of the developed products.

1.4 Research Questions

The research question motivating this thesis is: *How can we improve support for model evolution in model-driven development of automotive embedded systems?*

In the following we present two main research questions that focus on a certain aspects of the motivating question above. We decompose these questions further to direct and focus the work.

1. How can development tools used for the development of automotive embedded systems be integrated and exchange tool data?
   a) How can models be exchanged seamlessly between different tools?
   b) How can the data integration in a model-based tool integration solution be modularized and decomposed?
   c) What is the state of the art in model transformation technology?
   d) How can appropriate model transformation technology be selected and applied effectively in a model-based tool integration solution?
2. How can design decisions be documented in model-driven development?

a) What is the state of the art in design rationale and design decision documentation?

b) How can design decisions be represented in model-driven development?

c) How can the documentation of design decisions be linked to the model elements affected by this decision?

d) How can the double effort of both documenting design decisions and changing the model be reduced?

e) How can design decisions be documented in models conforming to an arbitrary metamodel, without changing the metamodel?

1.5 Summary of Contributions

In this section we summarize the contributions of this thesis according to the research questions raised in section 1.4.

RQ 1: How can development tools used for the development of automotive embedded systems be integrated and exchange tool data? We describe a method for modular, model-based data integration. We identify the different concerns addressed by data integration and use the principle of separation of concerns to decompose the data integration into separate modules: the technical space bridge and the structural bridge. For each bridge we select an appropriate model transformation technology. The basis for this selection is the survey of model transformation technologies. We apply this approach by integrating three development tools for automotive embedded systems: A systems engineering tool, a safety engineering tool and a simulation tool.

RQ 2: How can design decisions be documented in model-driven development?

We systematically survey and classify current approaches for design rationale and design decision documentation. We propose a design decision representation that is targeted to model-driven development. The design decision representation links documentation and model, lowers capture overhead through partial automation, ensures consistency between model and documentation through automation and is independent of a particular metamodel. We show in a case study how this representation can be used to document the stepwise refinement of models.

1.6 Thesis Outline

The rest of this document is structured as follows. Chapter 2 provides a brief overview of the state of the art in model-based and model-driven approaches, software architecture description, model-based tool integration and design decision documentation. Chapter 3 presents the research approach we took in this work, structured according to the two methodological research frameworks. Chapter 4 provides a summary of the approach and results of this work. In chapter 5 we discuss the

implications of this work and review the work. Chapter 6 presents a plan for future work. Chapter 7 concludes this document with a summary.

Chapter 2

State of the Art and State of Practice

The thematic map in figure 2.1 visualizes the areas of relevance for this work. For each of these areas, we provide an overview of the state of the art and the state of practice. In the later chapters of this thesis, we will use this map to place our work.

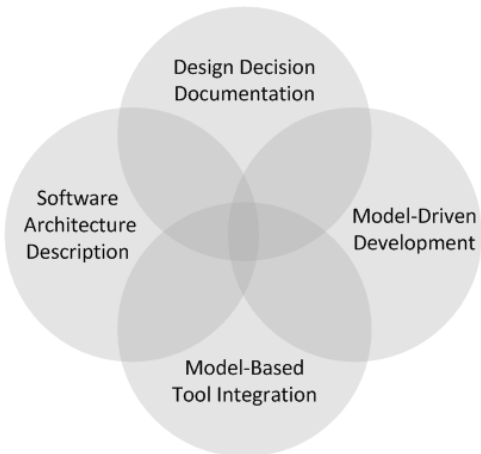

Figure 2.1: Thematic map

2.1 Model-Based and Model-Driven Approaches

Model-based and model-driven approaches use explicitly represented models as primary artifacts for describing a system [164]. During development, a series of such models is created, specified, refined and transformed. The models support various development activities, such as requirements engineering, architecture design, detailed design, analysis, simulation, implementation, testing and verification. The goals of these approaches are the management of complexity, the reduction of the risks of development, the improvement of the quality of the developed system and the improvement of the development efficiency.

Model-based and model-driven development appear in the literature under multiple names [161]:

- *Model-based* approaches focus on models as important artifacts for describing a system. Model-based engineering (MBE) and model-based systems engineering (MBSE) employ models for various engineering activities. Model-based development (MBD) is a specialization of MBE with a focus on development activities.

- *Model-driven* approaches are a specialization of model-based approaches. Models are the central artifacts that drive the development. This means that models are not just descriptive documentation, but they are used as construction tools. Model-driven approaches are commonly referred to as model-driven engineering (MDE), model-driven development (MDD) and model-driven software development (MDSD).

- Model Driven Architecture (MDA) is a recommended practice for MDD proposed by the Object Management Group (OMG) [165], where a *Platform Independent Model (PIM)* is transformed into a *Platform Specific Model (PSM)*. A PSM contains information about the chosen platform, a PIM is independent of a particular platform and describes the system in more general terms. A platform is a relative concept and can be used on several levels.

2.1.1 Terminology

Model: "A model is a simplification of a system built with an intended goal in mind. The model should be able to answer questions in place of the actual system" [17]. Several kinds of models exist [161]:

  - Mental models are used by individuals for understanding and sense making.
  - Conceptual models are used for documenting and communicating.
  - Formal/mathematical/analytical models are used to clarify and solve engineering problems and to make predictions of system properties.

- Constructive/executable models are used to specify detailed solutions, allowing partial automation.

The models used in model-driven development are usually conceptual and constructive, since they serve both as documentation and specification. Different approaches for representing models exist [84]:

- Domain-Specific Modeling Languages (DSML) capture the concepts in a certain domain and typically have a narrow scope [122, 153].
- General purpose modeling languages have a broad scope and are often standardized. An example is the Unified Modeling Language (UML) [170].
- Customizations of a general purpose modeling language are a compromise between the above options. An example is the UML profile mechanism, a lightweight extension for UML [170].

Model Elements: Models consist of several related *model elements*.

Metamodel: A metamodel of a model X describes the abstract syntax that model X must follow to be valid. A metamodel can be compared to a grammar in language design. Precisely defined metamodels are a prerequisite for model transformations [151].

Metametamodel: A metametamodel of model X is the model describing the metamodel of model X [80]. It can be compared to the grammar of the language that is used to describe the syntax of the language X (e.g. BNF [129]). Standards and established implementations for metametamodels exist, such as MOF [166] and Ecore [194].

Metamodeling Levels M0-M3: The metamodeling levels are defined by the OMG [166]. They describe the conformance relation between objects of the real world (M0), models (M1), metamodels (M2) and metametamodels (M3), as depicted in figure 2.2: a model conforms to a metamodel and a metamodel conforms to a metametamodel. Metametamodels can be used for their own definition, a concept called meta-circularity [92].

Model Transformation: Model transformation is the "automatic generation of one or multiple target models from one or multiple source models, according to a transformation description" [152]. The model transformation description is expressed in a model transformation language, such as QVT [168] or ATL [116]. To produce the target model, the model transformation description is interpreted by a model transformation engine. A model transformation description can be modeled as well, so the model transformation description is conform to a model transformation metamodel (see figure 2.2). In the appended paper A we review the state of the art in model transformation, including the terminology, usage scenarios, a classification scheme of the

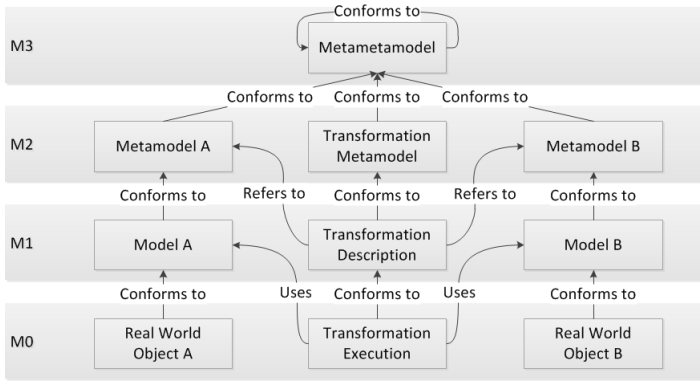

Figure 2.2: The metalevels of models and model transformations

problems that are solved by model transformations, a classification scheme of model transformation languages and an overview of several model transformation languages and engines.

Syntax: The syntax describes how model elements may be composed to form valid models. We can distinguish between abstract and concrete syntax. The abstract syntax is a description of the structure that is independent of any particular representation or encoding. The concrete syntax is a mapping of an abstract syntax onto a particular representation or encoding.

Semantics: The semantics defines the meaning of metamodel elements. Meaning is usually defined by mapping metamodel elements to concepts that already have a defined meaning.

Technical Space: The technical space (also called technological space) of a model concerns the technology used for representation of the model [152, 137, 80]. Examples for technical spaces are EMF (Eclipse Modeling Framework) [194] or XML (Extensible Markup Language) [32]. Each technical space defines a metametamodel and a set of tools and technologies to work with models, metamodels and metametamodels.

Level of Abstraction: A *level of abstraction* is a property of a model describing the amount of information contained in the model. While every model is an abstraction of the real world, a level of abstraction is defined by the amount of different questions that can be answered by a model. A model of a high level of abstraction answers less questions than a model of a low level of abstraction. Support for models of several levels of abstraction provides a mechanism for dealing with complexity. The term *level of abstraction* suggests that abstraction is discretized into several levels or steps (see section 2.2.3 for an example).

2.1.2 Model Evolution

Software evolution is a part of the software development life cycle [139] and deals with changes in software. The changes performed during software evolution have been classified into perfective, adaptive and corrective changes [143]. This classification has later been refined into evaluative, consultive, training, updative, reformative, adaptive, performance, preventive, groomative, enhancive, corrective and reductive changes [45].

Model evolution is a specialization of general software evolution; in the following we describe the aspects that are particular to the evolution of models. We classify model evolution by two orthogonal dimensions with each two categories and present the resulting 4 classes in table 2.1.

Models can be described by their information content and their syntax, which is defined by a metamodel. Model evolution describes and manages changes in both aspects:

- Content-related model evolution: The information content of models is changed, when model elements are added, modified or deleted.
- Syntactic model evolution: The metamodel (or abstract syntax) of a model is changed, when metamodel elements are added, modified or deleted.

Changes may affect the whole model or only a small part of the model. Model evolution can capture changes of different extent:

- Local model evolution: A subset of the model is affected by the change.
- Systemic model evolution: The whole model is affected by the change.

	Content-related	Syntactic
Local	(1) Adding/deleting/modifying of model elements e.g. [3]	(2) Co-evolution of metamodels and models e.g. [182]
Systemic	(3) Differencing/merging of models e.g. [47]	(4) Translation/synchronization of models to conform to other metamodels e.g. [4]

Table 2.1: Classification of model evolution

In the following we provide some examples for using the characterization above:

- Local content-related model evolution: A new model element is added to a model.
- Local syntactic model evolution: The co-evolution of models and metamodels deals with the automatic evolution of models to cope with a given change in the metamodel [102, 100, 101, 218, 158].

- Systemic syntactic model evolution: A complete model is translated to conform to another metamodel.

2.1.3 State of Practice

The complexity of embedded systems puts heavy requirements on the development process that is used to build these systems. In addition, the improvement of efficiency and reduction of the cost of embedded systems development is a major concern for industrial practice [5, 6, 68].

In the industrial development of automotive embedded systems, document-based approaches dominate and model-driven approaches are locally adopted in well-delimited projects, forming islands of model-driven development. A recent study claims that 37 % of embedded systems are developed model-based [26]. Model-driven and model-based approaches promise to manage the complexity inherent in the development of embedded systems, to improve efficiency, improve quality and to provide an early analysis and simulation of various system properties [69]. There is still no generalizable evidence that model-driven development fulfills the promises it makes in large-scale industrial applications [155]. Experiences with industrial case studies in the MODELWARE project report mixed results: While some projects report an improvement of efficiency, other projects report decreased efficiency due to insufficient tool support [154].

Support for long-term model evolution is especially relevant, when models have along lifetime. Automotive embedded systems have a development phase of 4-6 years, production phase of 5-6 years and an aftermarket support and service phase of 10-15 years. In a timespan of over 20 years the models of the embedded system change and evolve, a process which is not well supported today [35, 178, 184]. The four classes of model evolution identified in the previous section impose challenges on the use of model-driven development in practice. Dedicated model management systems are rarely applied, instead generic version management systems are used for modeling data.

2.2 Software Architecture Description

The architecture of a software-intensive system, such as an embedded system, is defined as “the fundamental organization of a system embodied in its components, their relationships to each other and to the environment, and the principles guiding its design and evolution” [110]. The architecture captures the early and important design decisions of the system under development [64]. The terminology and best practices regarding software and systems architecture are specified in the two standards for architectural description of software-intensive systems (IEEE/ANSI 1471-2000) [110] and ISO/IEC 42010:2007 [111].

2.2.1 Views, Aspects and Consistency Checking

An architectural *view* is a representation of the system from the perspective of a related set of concerns [49, 110]. An architectural view provides a coherent selection and projection of the architecture according to the interests of a specific stakeholder. Views are intended to reduce the perceived complexity for a specific stakeholder. A *viewpoint* establishes the purpose of a view and the techniques or methods used for constructing the view [183, 110]. The viewpoint also specifies the conventions for interpreting, constructing and using an architectural view.

The concept of aspect-orientation is related to the concept of views. Aspect orientation was first introduced as Aspect-Oriented Programming (AOP) [128] and has been extended to Aspect-Oriented Modeling (AOM) [83] to consider aspects on a higher level of abstraction than programming. Aspect-orientation applies the principle of separation of concerns to identify, isolate and localize crosscutting solutions. These crosscutting solutions are described separately in aspect models and a primary model. An integrated system model is created by weaving, i.e. composing selected aspects with the primary model.

Views and aspects are both approaches to reduce the complexity of a complete system model by limiting or filtering the presented information. Views are usually only loosely related, whereas aspects are explicitly coupled and woven with each other.

Different views describing the same system need to be consistent. If inconsistencies between views exist, they need to be detected and resolved. Consistency between two descriptions is defined as the coexistence of the two descriptions without contradiction. We can differentiate two types of consistency checking: (1) Consistency checking between artifacts of the same stage of the development process might for example check the consistency between UML state machines and UML sequence diagrams [150, 27]. (2) Consistency checking of artifacts from different stages of the development process might for example check the consistency between a class diagram (design stage) and source code (implementation stage) [156, 131, 46, 22, 177, 70, 71, 162, 157]. Instead of checking for consistency, other approaches have the goal of enforcing consistency, such as tool and data integration approaches (c.f. section 2.3).

2.2.2 Architecture Description Languages

Architectural Description Languages (ADLs) describe and formalize a high-level decomposition of a system in terms of structure and behavior. The essential elements of an architecture described by ADLs include components, connectors, interfaces, and their behavior [148]. ADLs can be used to perform analysis, verification, reasoning and quality assessment (e.g. completeness, consistency, performance, safety) early in the development process, when changes are comparably simple and cheap.

A large number of ADLs have been introduced in the 90s, such as AADL/MetaH (by SAE and SEI), C2 (by UCI), Wright (by CMU), ACME (by CMU), Koala/-

Darwin (by Imperial College and Philips) and Rapide (by Stanford). Clements [50] describes the languages in an early survey. Medvidovic et al. [148] compare the language constructs provided by ADLs. El-khoury et al. [74] compare architecture description languages specifically for embedded systems. Hill et al. [105] compare the languages by their different levels of expressiveness.

Some of the before mentioned ADLs are not in use any longer. Medvidovic explains the short lifetime of some ADLs with their focus on technological aspects and with their lack of consideration of the application domain, business context and development context. As a result, Medvidovic proposes domain-specific ADLs [147] focusing on a particular domain and business. Another report on the use of ADLs in practice stresses that ADLs need to align more closely with industrial software architecture practice [224].

The ADL approach can be combined with model-based and model-driven approaches. In this case, ADLs can be implemented as domain specific modeling languages (DSMLs) [122] or as profiles of the Unified Modeling Language (UML) [170].

2.2.3 EAST-ADL2 - An ADL for Automotive Embedded Systems

Problems in interdisciplinary communication often arise from the lack of a shared terminology [2]. A language with well-defined semantics that several disciplines can relate to establishes a common ground for interdisciplinary communication. EAST-ADL2 is an architecture description language for the domain of automotive embedded systems [59, 57]. In comparison to earlier architecture description languages (c.f. section 2.2.2), EAST-ADL is specific to one domain, the automotive domain, and is aligned with industrial practices and standards such as AUTOSAR and ISO 26262.

EAST-ADL2 can be used to describe high-level abstract functions and both the computer hardware and the software architecture of an automotive embedded system. An EAST-ADL2 model comprises four levels that describe the same system from different viewpoints and on different levels of abstraction (see figure 2.3).

- The *Vehicle Level* describes the system on high level of abstraction and contains a model of the electronic features of a car. A product line architecture can be realized using a hierarchical variability mechanism, feature modeling and feature configuration [179].

- The *Analysis Level* contains the Functional Design Architecture, which describes the system in terms of functions and their connection to the environment through sensors and actuators.

- The *Design Level* contains the *Functional Design Architecture* and the *Hardware Design Architecture*. The *Functional Design Architecture* describes the software functions as a decomposition of the functions on analysis level. The

Hardware Design Architecture describes the topology of the hardware. In addition, the allocation of software functions to hardware components is described.

- The *Implementation Level* contains an AUTOSAR-conform implementation, where the functions of EAST-ADL2 are mapped to AUTOSAR Runnables.

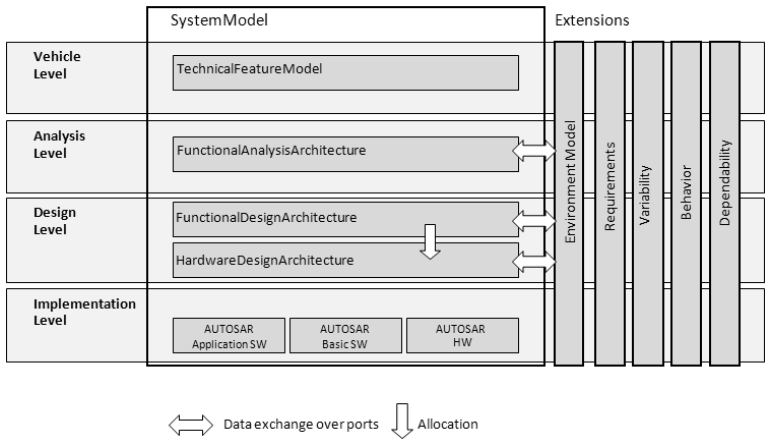

Figure 2.3: EAST-ADL2 abstraction levels

Each involved discipline traditionally has its own view on the architecture and may prefer to work with a specific abstraction level of EAST-ADL2. Control engineers may prefer to work with a functional view of the system, provided by the analysis level, whereas software engineers may prefer the component-oriented view provided by design level and implementation level.

Several language extensions for EAST-ADL2 exist, focusing on timing [203], environment modeling, behavior modeling, variability modeling, error modeling and requirement modeling. The EAST-ADL2 metamodel is implemented as a UML profile based on constructs inherited from UML [170], SysML [169] and AUTOSAR. Using the EAST-ADL2 profile for UML, it is possible to create and edit EAST-ADL2 models in UML design tools. EAST-ADL2 complements *AUTOSAR (Automotive Open Software Architecture)* [8] by providing a metamodel for describing the system at a higher level of abstraction.

EAST-ADL2 has been developed collaboratively between industry and academia in the European research projects EAST-EEA, ATESST and ATESST2. In this work, EAST-ADL2 is used as the basis for our case-studies.

2.2.4 State of Practice

The software architecture in automotive embedded systems is growing in size and complexity, about 80% of all new functions introduced in cars today are software-based [184]. At a high-level of abstraction, *reference architectures* are used to describe the strategic and technical aspects of a vehicle, including product line strategies for reuse and the high-level decomposition of software and hardware of a vehicle, such as ECUs (Electronic Control Unit), communication buses and logic partitioning of functionality [178]. Reference architectures are usually provided in the form of text and schematic drawings. Functionality is decomposed and modularized into components. However, there is no traceability between the high-level reference architecture and the low-level component implementation.

AUTOSAR (Automotive Open System Architecture) [8] is an industry initiative to create a standard framework for the automotive software architecture including a component model and middleware platform. AUTOSAR decouples hardware and software through several abstraction layers and also standardizes the interfaces between different software components. As the interfaces are restricted to ensure interoperability, competitors can differentiate in functionality and implementation. In industrial practice, AUTOSAR is currently introduced in some components, partly for newly developed functionality, partly for migrated, existing legacy software [135].

2.3 Model-Based Tool Integration

The models used in model-driven approaches can be diverse, as they represent different views of the system with respect to the involved domains or lifecycle phases. The different models are supported by dedicated tools for editing, analyzing, simulating, report generation or code generation. These tools are often isolated and do not interact or exchange information.

Tool integration is concerned with the relationships among tools, the properties of the relationship and the degree to which tools agree [201]. The goal of tool integration is building an engineering environment of several development tools [34]. The assumption of the research community is that an engineering environment with several integrated tools increases productivity and product quality [221]. A broad overview of the literature on tool integration is provided in the annotated bibliographies of Brown [34] and Wicks [222].

The scope of tool integration needs to be separated from the mechanisms used. We introduce the *mechanisms* for model-based tool integration in section 2.3.1. The *scope* of tool integration is defined using the following dimensions [219, 201]:

- *Data integration* shares the data produced by different tools and manages the relationship between the data objects.

- *Control integration* allows tools to notify and activate other tools.

- *Presentation integration* provides a common user interface with a common look-and-feel.
- *Process integration* provides process management tools with data from development tools.
- *Platform integration* provides a virtual operating environment for heterogeneous hardware and software.

A specific kind of tool integration approach is *model-based tool integration*, which uses modeling technology, such as metamodels and model transformations, to realize data integration solutions. The tool data is stored in tool-specific models, which correspond to tool-specific metamodels. Model-based tool integration provides means for specifying and mappings between tool-specific metamodels to overcome semantic, syntactic and technical heterogeneities between the tools [119]. Tool integration technology is applied to build *tool chains*. A tool chain is a collection of tools that are integrated and interoperate seamlessly, so the output of one tool becomes the input of the next tool in the tool chain.

2.3.1 Approaches for Data Integration

In the following we focus on the dimension of data integration. Several mechanisms for realizing data integration using modeling technology have been proposed [119]. In this section we will introduce them briefly.

Tool Integration Architectures

There are two high-level architectures for data integration, as identified by Karsai et al. [120]. The first approach is based on point-to-point bridges between tools. The challenge is the number of bridges required for integrating multiple tools, as each tool needs to connect to each other tool. The second approach is based on a central repository and a common metamodel. Each tool requires only a single bridge to connect to the repository. The challenge of this approach is defining the information contained in the common metamodel.

Ontologies and Information Models

An ontology is defined as a "formal, explicit specification of a shared conceptualization" [95]. This shared conceptualization provides a common reference for several tools [163, 217]. An example of a model-based tool integration approach using ontologies is the ModelCVS project. It maps tool-specific metamodels to an ontology and defines relationships between the tool-specific ontology elements [118, 117, 132].

Related approaches establish a common information metamodel for a specific domain, such as EAST-ADL2 [57] for the automotive embedded systems domain

and the CESAR Common Metamodel [13] for the embedded systems domain. Herzog [104] proposes a related approach for the exchange of systems engineering data using a common information model.

Model Weaving and Aspect Orientation

Composition based approaches focus on combining aspects from different models [16, 48]. Aspect oriented approaches weave aspect models that focus on a specific concern into a comprehensive model [83]. Aspects models are combined according to a weaving model, which specifies how models can be combined. Technologies for model weaving are XWeave [94] and AMW [61, 62, 60].

Model Transformations

Model transformations have a central role in model-based and model-driven engineering, as they can be used to automate the creation and adaptation of models. Model transformation technology can be used for data integration, where model transformations describe mappings between tool-specific metamodels [208]. Several approaches explore the use of model transformation languages for data integration, for example the MOFLON language [4] and Eclipse-based languages [72]. Model synchronization approaches are a subgroup of model transformation approaches that have been used for data integration. Model synchronization enforces consistency between two or more models [93]. A change in either of the models triggers an appropriate, incremental update of the other model.

Integration Languages

When mappings between metamodels are defined, patterns of mapping rules can be identified that recur frequently. Integration languages leverage this knowledge by providing specialized language constructs for each integration pattern [88]. The CAR mapping language [180] is an example for an integration language based on patterns.

Automated Mapping Generation using Heuristics

Mappings between two metamodels can be computed automatically under the assumption that the metamodels are similar. So called *matching algorithms* compute mappings between two given metamodels. The approaches are heuristics, i. e. follow a best-effort strategy using metrics. Relevant metrics are structural similarity of the metamodels [79], naming similarities between metamodel elements or a combination of different metrics [63]. A domain specific language for specifying matching algorithms has been proposed [88].

Model Exchange Formats

Several standards for model exchange formats have been proposed. There is the format XMI (XML Metadata Interchange) [167] defined by the OMG and the STEP Standard (ISO 10303) [113] with several application protocols. A relatively new initiative are the Open Services for Lifecycle Collaboration (OSLC or Open Services) [171, 223]. The initiative proposes standardized definitions for resource and interface descriptions for sharing engineering artifacts such as requirements.

These standards and initiatives define the low-level structure of the elements to be exchanged, however, they do not define the mapping of data to existing tools. The standards are quite general and do not take domain-specific information into account.

Data Management Approaches

Data Management approaches provide a framework for data integration. However, they do not define how the data between specific tools is actually mapped. Also, each of the different classes of data management approaches are growing increasingly comprehensive in both scope and functionality. This results in an assimilation of the traditionally separate approaches, where a clear distinction is no longer apparent in all cases. In the following we present different types of data management approaches.

- *Model Management Platforms* manage data in the form of models. They provide a model repository, model registry, model transformation tools, editing tools and browsing tools. Model management also provides operations on models, for example `match`, `diff`, `copy`, `merge`, `compose`, `invert` [14]. A model registry is provided for identifying, storing and retrieving both models and metamodels. An example of an early model management platform is Rondo [149]. It is increasingly recognized that not only the model, but also the mapping between different models needs to be managed by these platforms [15], resulting in an assimilation of model management platforms and integration platforms.

- *Integration Platforms* provide a central data repository and several tool adapters. ModelBus [98] and Jazz [85] are examples of service-oriented integration platforms that provide storage and version management. They use web services, which can be orchestrated. The web services implemented in Jazz use a RESTful (Representational State Transfer) [81] architecture and conform to the specification of the OSLC initiative (Open Services for Lifecycle Collaboration) [171, 223].

- *Software Change and Configuration Management (SCM)* tracks and controls the changes in software source code that is developed in distributed settings [10]. In general, the integration aspects of SCM are limited to a common

database or repository for software artifacts and operations such as `diff` and `merge`. Examples of SCM systems are the Concurrent Versioning System (CVS) [214] and Subversion (SVN) [176].

- *Product Data Management (PDM)* integrates product-related data of mechanical products and associated work-flow processes. It serves as a central repository for the creation, management and publication of process and product history. It provides version management, change management and configuration management. An integration of PDM and SCM systems has been studied [55].

- *Product Lifecycle Management (PLM)* manages the product-related data for the entire life cycle from the conception of the product, through design and manufacturing, to service and disposal.

- *Application Lifecycle Management (ALM)* integrates data from different software development activities and life cycle activities such as requirements management, architecture, coding, testing, tracking, and release management.

2.3.2 State of Practice

The development of embedded systems is a multi-disciplinary engineering effort. Engineers of a number of different disciplines need to contribute with their expertise in the development of automotive embedded system. Each of the disciplines uses its own set of tools, languages and metamodels to describe the embedded system from a disciplinary perspective [73, 127]. The views created by the different tools might contain redundant and inconsistent information. Manual translation, synchronization and updating of views lead to development inefficiencies. The lack of interoperability leads to gaps in the development process, high development costs and reduced product quality [185].

Especially the maturity of tool environments for model driven development and the lack of interoperability among tools are perceived as a hindrance for large-scale industrial adoption of model-driven development [11]. A recent survey on the industrial use of tools in embedded systems development shows the lack of tool integration, traceability and documentation [112]. Islands of tool integration exist, for example between modeling tools and source code tools: UML modeling tools allow round trip engineering between models and source code. MATLAB/Simulink allows automatic code generation from models. In addition, a number of ad-hoc integration solutions for specific tools exist, often custom-created out of necessity for a specific company.

A systematic integration of tools is desirable to reduce inconsistencies and increase the efficiency of development. However, the tools do not interoperate well and are often not designed to interoperate. There is a lack of systematic approaches for tool integration [204].

2.4 Design Decision Documentation

Design decisions are the far reaching decisions that have an impact on the architecture and its description. Design decisions are usually made in the early phases of development and are hard to change later. *Design rationale* captures the reasoning underlying the creation and use of artifacts, such as models and model elements, and the reasoning underlying the design decisions [38]. Whereas the software architecture models focus on the *"what"*, design rationale and design decision documentation focus on the *"why"*.

A number of approaches for representing design decisions exist. These different representations have varying degrees of formality and rigor. Informal representations document design decisions in the form of free text, use cases and videos [43]. Template-based approaches provide a guideline for textual descriptions of design decisions. An example is the template by Tyree and Akermann [209].

The majority of approaches structure design decisions according to a metamodel. These approaches explicitly represent the design deliberation process, including alternatives and arguments. Early approaches for design rationale and design decision representation are IBIS (Issue-Based Information Systems) [136], PHI (Procedural Hierarchy of Issues) [146], DRL (Decision Representation Language) [138] and QOC (Questions Options and Criteria) [145].

A number of web-based approaches for architectural knowledge management have been proposed for capturing design decisions. Examples are PAKME [9], ADkwik [186] and ADDSS [41]. These tools capture information about design decisions in web-forms and store them in a database.

Approaches for representing design decisions that were developed in recent years focus on linking design decisions with the architecture description. The architecture and its design decisions are shown in the same environment. The SEURAT approach links design rationale to specific lines in the source code [37]. AREL is a UML profile that can be used to annotate architectural models with design decisions and rationale [199]. The Archium approach associates design decisions with modifications of the architecture [115]. These models provide descriptions, traceability and linking to other modeling elements.

Rationale provides information both to the rationale author and to other stakeholders [190, 191]. Rationale provides a memory aid for the rationale author and it is also a means for communicating architectural knowledge to other stakeholders [52, 51, 99]. Rationale can be used for change management to predict the impact of a change in the architecture [38, 200]. Rationale improves both efficiency and correctness of the change impact analysis [31]. Rationale can improve maintenance activities, as it improves the understanding of the system and reduces the time needed for maintenance tasks [121, 40]. Rationale can document the assumptions of system components, this allows the safe reuse of the components in new systems [142, 220, 96]. Falessi et al. [77, 78] investigate empirically, in which cases rationale is useful for developers. They design a controlled experiment to test the perceived value of design decisions for performing different software engineering activities.

One of their findings is that there are big differences in perceived value, depending on which architectural element the rationale is associated with.

The documentation of design decisions for safety-critical embedded systems systems has been investigated by Wu and Kelly [225] and by Leveson [141]. The lack of documentation of design decisions for embedded systems can lead to development delays, cost overruns, disruptions, upgrades and system evolution [140]. The documentation of safety-critical design decisions is on one hand intended to support the developers, on the other hand it is intended to be used for the certification of the product by authorities. The safety case concept of Kelly is targeted towards authorities and certification [123]. The intent specification [142] by Leveson is targeted towards developers. It organizes the information in a number of hierarchical views: the management view, customer view, system engineering view, component designer view, component implementer view and operations view. The views represent different levels of intent. Intent specifications have been used to document assumptions, rationale and traceability of software components to enable their safe reuse [220, 160].

Paper D provides an extended overview of the state of the art in design decision documentation.

2.4.1 State of Practice

Tang et al. [198, 197] study empirically how software professionals work with design rationale and design decision documentation. He found that software professionals recognize the importance of rationale for their work; however, they do not use dedicated tools for capturing and retrieving them. He also found that architects often omit to document their design decisions due to the involved overhead in time and effort. As a result, this knowledge about the architectural model is lost, a phenomenon known as *knowledge vaporization* [114].

While some dedicated representations and tool support for capturing design decisions have been proposed by academia, the capturing of design decisions and their rationale still creates additional overhead. The overhead caused by capturing design decisions is the main hindrance for consistent design decision documentation in practice [134].

The quality of documentation has a large effect on the quality of software [215]. Empirical studies have shown that inconsistent, outdated or missing documentation is a major cause of defects in development and maintenance [42, 174, 181, 53]. Documentation and architecture are often kept separate and evolve separately, so if the architecture changes, the technical documentation is not updated appropriately. The resulting separate documentation is an unreliable source of information [216]. This may result in inconsistencies, design erosion and architectural drift [207, 173, 175]. Maintenance accounts for 67 - 90 percent of the total cost of the software system [76, 108, 67, 226]. Maintainers spend most of their time understanding the existing system before they are able to change it. Design rationale and decision

documentation has the potential to improve the maintainer's understanding of the system [40].

The models used in model-driven development capture the outcome of the design process. The models do not explicitly capture the design decisions and reasoning behind the models. Currently, it is most common to complement the models with separate text or spreadsheet documents [112] that provide additional information regarding the models such as design decisions and design rationale. It is desirable that model-based development also supports metadata aspects such as design decisions [1] and to link the design decision documentation consistently with the model.

2.5 Chapter Summary

In this chapter we provided a brief overview of the state of the art in model-based and model-driven approaches, software and systems architecture, model-based tool integration and design decision documentation. Additional in-depth state of the art surveys can be found in appended paper A on model transformation technology and in appended paper D on design rationale and design decision documentation.

Chapter 3

Research Methods

In the natural sciences, the research method usually comprises proposing a hypothesis, performing a controlled experiment and accepting or rejecting the hypothesis based on the outcome of the experiment [44]. Embedded systems development is a relatively young, multi-disciplinary field. Researchers use a number of research methods, for example case studies [82], field studies, literature studies, simulations, action research, proofs [65, 227], surveys, empirical studies, controlled experiments [202, 227, 12, 66].

Methodological frameworks are used to provide a high-level structure for the chosen research approach. Since research in embedded systems is multidisciplinary, we present two methodological frameworks from different fields relevant for research in embedded systems development: Shaw's framework for software engineering research and Blessing's framework for engineering design research. We apply both of the frameworks to structure the research approach presented in this thesis.

3.1 A Methodological Framework for Software Engineering Research

In general, software engineering seeks better ways to develop and evaluate software, motivated by practical problems, with key objectives being quality, cost and timeliness. Shaw [189] proposes a framework for describing a research approach in software engineering. She decomposes the research approach into research questions, research results and research validation. She then defines a classification for the different techniques that can be used to implement these components.

- Research Question: To establish what is interesting to be researched, *research questions* are put forward. Typical research questions in software engineering can be classified according to their topic into methods of development, methods for analysis, design and evaluation of a particular instance, generalization or characterization schemes and feasibility studies.

- Research Results: The answers to research questions are research results. Typical research results in software engineering can be classified into procedures or techniques (new or better ways to solve a problem), qualitative or descriptive models (taxonomy for a problem area, checklist), empirical models, analytic models (structural models for formal analysis and automatic manipulation), notations or tools (formal language and tool support for this language), specific solutions, judgments and reports.

- Research Validation: The research results need to be validated. Validation can be accomplished through experience with the newly developed program or through systematic analysis [189]. The list of validation techniques used in software engineering includes case studies, formal analysis, empirical models, experiments and experience (the program is used by someone else and deemed correct, useful or effective). We add to Shaw's framework the distinction between internal and external validation. Internal validation, also called verification, checks that the proposed solution fulfills the requirements, whereas external validation checks that the proposed solution fulfills the expectations.

3.2 A Methodological Framework for Engineering Design Research

Engineering design research has two objectives, the development of models and theories of design and providing tool support [28]. Blessing [28] proposes a design research methodology (DRM) as a framework to provide methods and guidelines for doing research in design. The DRM framework is composed of the following stages:

- Research Clarification: The goal of this stage is to identify the topics of relevance and contribution, clarify the current understanding, clarify the research questions and create a research plan.

- Descriptive Study 1: The goal of this stage is to gain an in-depth understanding of the existing situation, e.g. by a literature survey.

- Prescriptive Study: The goal of this stage is to create support for improving the existing situation.

- Descriptive Study 2: The goal of this stage is to evaluate the support developed in the prescriptive study.

3.3 Research Methods in this Thesis

In the previous sections, methodological frameworks for research in software engineering and engineering design have been introduced. In this section we apply these frameworks to describe the research work presented in this thesis.

Table 3.1 presents the research in this thesis according to the framework proposed by Shaw [189]. For each tackled research question, we identify the type of research question, the type of research result and the type of research validation.

Research Question	Type of Research Question	Type of Result	Type of Validation
RQ 1	Method of Development and Particular Instance	Method and Tool	Case Study
RQ 2	Method of Development and Feasibility Study	Descriptive Model, Notation and Tool	Case Study

Table 3.1: Classification of research questions, results and validation in this thesis according to section 3.1

In the following we apply the design research methodology introduced in section 3.2 to describe the research in this thesis.

- Research Clarification: The research questions are refined into several more precise and more concrete questions.
- Descriptive Study 1: The study of the existing situation and related approaches is done by literature study.
- Prescriptive Study: The proposed improvements include a tool integration solution and a representation for documenting design decisions.
- Descriptive Study 2: The proposed improvements are evaluated by examples and case studies by coming back to the research question.

We arrange the steps in a circular way (see figure 3.1), to illustrate that several versions are developed and with each version we apply the learning from previous experiences. There are three such circles in this work. The first circle describes our approach for tool integration. The second circle describes the approach for design decision documentation. The third circle describes future work (see chapter 6). It uses the previous two circles as a foundation and combines the areas of tool integration and design decision documentation into an approach for exchanging models together with their documentation between different tools. In figure 3.1 we provide a reference to the section that describes the respective step.

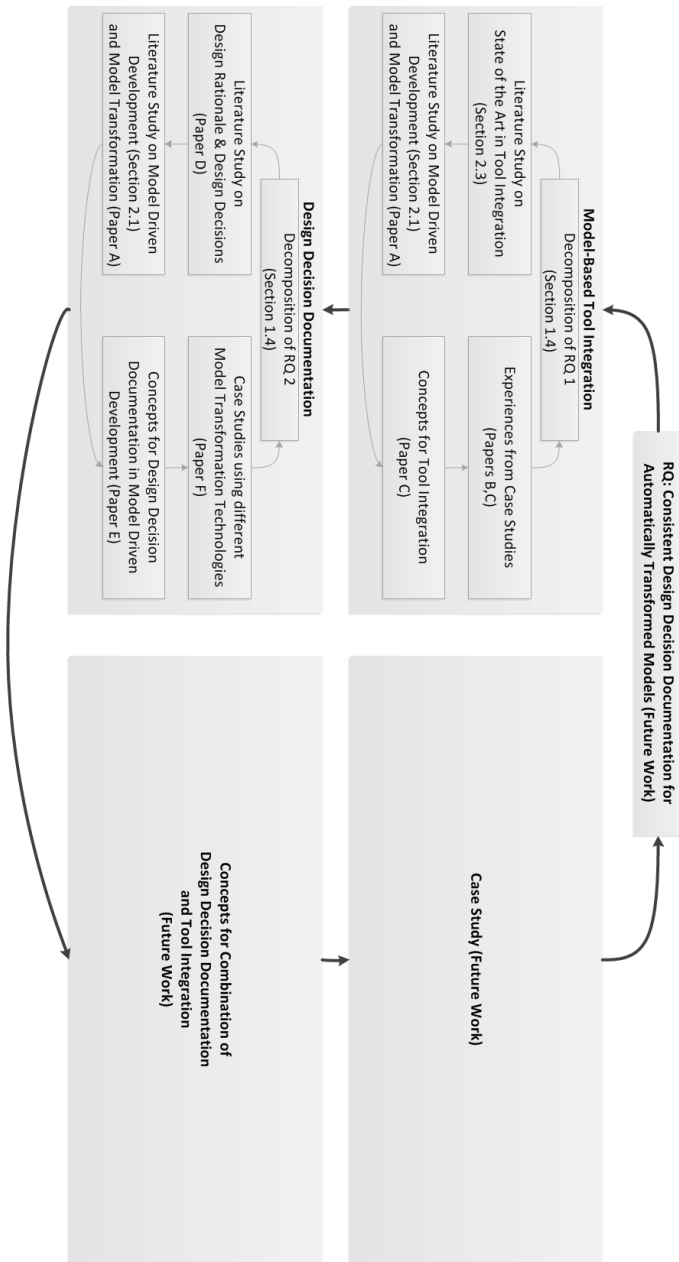

Figure 3.1: Overview of the research approach

3.4 Chapter Summary

To cover the scope of the multidisciplinary topic of research in embedded systems development, two methodological frameworks are presented in this chapter, one for software engineering research and another for engineering design research. We use these frameworks for structuring the research approach presented in this thesis.

Chapter 4

Approach and Results

In this chapter we present the approach and results of this work, structured according to the research plan outlined in figure 3.1 on page 32. In the following we summarize the work for each research question and relate it to the appended publications.

The publications appended to this thesis are in the fields of model-driven development, tool integration and the documentation of design decisions. Figure 4.1 visualizes the placement of the appended publications on the thematic map. In the following we briefly present the main points of each paper.

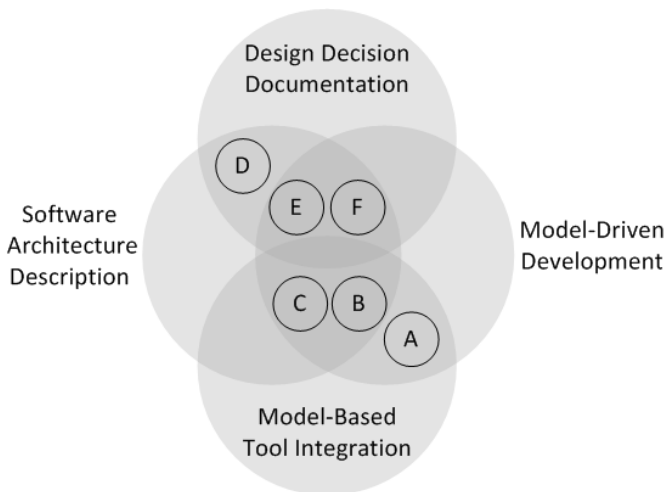

Figure 4.1: Appended publications on the thematic map

Paper A

Matthias Biehl, "Literature Study on Model Transformations," Technical Report, Royal Institute of Technology, ISSN 1400-1179, ISRN/KTH/MMK/R-10/07-SE, Stockholm, Sweden, July 2010.

Paper A surveys the state of the art in model transformation technology.

Model transformation is a central concept in model-driven development approaches, as it provides a mechanism for automating the manipulation of models. In this document we survey and classify existing model transformation technology. The classification differentiates between the problem space, i.e. characteristics of the problem to be solved by model transformation technology, and the mechanism, i.e. characteristics of the model transformation languages and engines. We show typical usage scenarios for model transformations and identify characteristics of the problems that can be solved with the help of model transformations. We synthesize a unifying classification scheme for model transformation languages based on several existing classification schemes. We introduce a selection of model transformation tools available today and compare them using our classification scheme.

Paper B

Matthias Biehl, Chen De-Jiu, and Martin Törngren, "Integrating Safety Analysis into the Model-based Development Toolchain of Automotive Embedded Systems," in *Proceedings of the ACM SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems (LCTES 2010)*, April 2010, pp. 125+.

Paper B presents a tool integration approach for safety analysis using model transformations.

The automotive industry has a growing demand for the seamless integration of safety analysis tools into the model-based development tool chain for embedded systems. This requires translating concepts of the automotive domain to the safety domain. We automate such a translation between the automotive architecture description language EAST-ADL2 and the safety analysis tool HiP-HOPS by leveraging the advantages of different model transformation techniques. By means of this integration, the safety analysis can be conducted early in the development process, when the system can be redesigned to fulfill safety goals with relatively low effort and cost.

Paper C

Matthias Biehl, Carl-Johan Sjöstedt, and Martin Törngren, "A Modular Tool Integration Approach - Experiences from two Case Studies," in *3rd Workshop on*

Model-Driven Tool & Process Integration (MDTPI2010), June 2010.

Paper C reports the lessons learned from two tool integration approaches using model transformations.

In the model-driven development process of automotive embedded systems a number of specialized tools are used to support various development tasks. Each tool needs to work seamlessly with artifacts created by other tools to increase the efficiency of development. We identify desirable properties for integrating the data of different tools. We then propose an approach for decomposing the data integration into modular steps that fulfill these properties. We report our experiences from applying this approach to integrate simulation capabilities and functionality for safety analysis into a model-based development environment.

Paper D

Matthias Biehl, "Literature Study on Design Rationale and Design Decision Documentation for Architecture Descriptions," Technical Report, ISSN 1400-1179, ISRN/KTH/MMK/R-10/06-SE, Royal Institute of Technology, Stockholm, Sweden, July 2010.

Paper D surveys the state of the art in design decision documentation and design rationale.

In this document we provide an overview of the state of the art in documentation of design rationale and design decisions for architecture descriptions. We define the terminology of the area and compare the concept of rationale to similar concepts. We provide an overview of areas of contemporary research in design rationale. For each of the identified areas, we describe both the challenge and proposed solutions. Based on the findings from the literature we present benefits and inhibitors for using rationale and design decision documentation.

Paper E

Matthias Biehl and Martin Törngren, "An Executable Design Decision Representation using Model Transformations," in *36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA2010)*, September 2010.

Paper E proposes a representation for the purpose of documenting design decisions in model-driven development using model transformation technology.

Design decisions are often tacit knowledge of an architecture and consequently they are easily lost during software evolution, a phenomenon known as knowledge vaporization. As a countermeasure, design decisions can be documented explic-

itly. However, documenting design decisions is expensive because they need to be captured in addition to the changes in the architecture. We propose an executable representation for design decisions using model transformations, which is independent of a particular component model or architectural description language. As a result, this provides the possibility to reduce knowledge vaporization. At the same time we prevent the high capturing cost, since the corresponding architectural change can be computed automatically.

Paper F

Matthias Biehl, "Documenting Stepwise Model Refinement using Executable Design Decisions," in *International Workshop on Models and Evolution (ME 2010)*, October 2010.

Paper F applies executable design decisions in a case study to document the stepwise refinement of a model.

During model refinement a wealth of knowledge about the model under development is accumulated that is only partly represented by the model itself. Design decisions and the considered modeling alternatives are neither represented by the model nor are they documented. During later lifecycle stages this information is often not available any more, which reduces the understandability of the model and potentially leads to inconsistencies and erosion of the model. We propose an approach to capture and store the design decisions in model-driven development. We represent design decisions as model transformations and propose tool support that applies this representation to capture design decisions with low effort. The captured design decisions provide a record of the model evolution and the rationale of the evolution.

4.1 Two Dimensions of Model Evolution

In section 2.1.2 we introduced model evolution and characterized it. To support the evolution of models, we need to consider how models are created[1] and changed. We visualize two different ways in which models can evolve in diagram 4.1.

- A model $M_{a,0}$ can evolve by adding, deleting or modifying model elements, resulting in $M_{a,1}$. This is depicted by the vertical arrow $d_{a,1}$ in equation 4.1. Both $M_{a,0}$ and $M_{a,1}$ correspond to the same metamodel N_a .

- A model $M_{a,0}$ can evolve by changing the metamodel it conforms to, resulting in $M_{b,0}$. This is depicted by the horizontal arrow t_{ab} in equation 4.1. Both $M_{a,0}$ and $M_{b,0}$ correspond to different metamodels N_a and N_b respectively.

$$\begin{array}{ccc} M_{a,0} & \xrightarrow{t_{ab}} & M_{b,0} \\ \Big\downarrow{\scriptstyle d_{a,1}} & & \\ M_{a,1} & & \end{array} \tag{4.1}$$

We will examine the two dimensions of model evolution in more detail.

- The horizontal dimension of model evolution captures the *systemic syntactical model evolution*, according to the terminology introduced in section 2.1.2. It involves changing the metamodel that the model conforms to, while at the same time preserving the semantics of the model. This aspect of model evolution is also called *model migration* [192] and in the area of tool integration it is represented by the dimension of *data integration* [219]. Horizontal model evolution is often found in model-based tool integration scenarios and can be practically realized as a *tool chain*, where different tools exchange the information contained in tool-specific models. We use tool bridges involving model transformations to describe this dimension. They allow us to compute a model for a certain tool-specific metamodel from a model corresponding to another tool-specific metamodel.

- The vertical dimension of model evolution captures the *content-related model evolution*, according to the terminology introduced in section 2.1.2. This includes operations on models, such as adding, deleting or changing of model elements. A design decision is made up of several such elementary operations. A sequence of several design decisions forms a *decision chain*. We use executable design decisions to describe this dimension of evolution. They allow us to establish a functional relationship between sequential versions of a model, and they allow us to compute a certain version of the model from a

[1] Model creation can be considered to be a special case of model change, where the initial model is empty.

previous version of the model. All versions of the model correspond to the same metamodel.

To clarify the two dimensions of model evolution, we describe them mathematically using algebra. Let metamodel N_a be the tool-specific metamodel of tool T_a. Let $M_{a,i} \in N_a, i \in \mathbb{N}$ be a model corresponding to metamodel N_a. The index i is the refinement number of the model resulting from applying design decision $d_{a,i}$. Let $d_{a,i} : N_a \to N_a$ be a design decision for a model of type N_a with $M_{a,i} = d_{a,i}(M_{a,i-1})$. Analogous definitions are made for tool T_b: Let $M_{b,i} \in N_b, i \in \mathbb{N}$ and $d_{b,i} : N_b \to N_b$ with $M_{b,i} = d_{b,i}(M_{b,i-1})$. We can now express any refinement $j \in \mathbb{N}$ of model $M_{a,j}$ and $M_{b,j}$:

$$M_{a,j} = (d_{a,j} \circ d_{a,j-1} \circ ... \circ d_{a,1})(M_{a,0}) \tag{4.2}$$

and

$$M_{b,j} = (d_{b,j} \circ d_{b,j-1} \circ ... \circ d_{b,1})(M_{b,0}) \tag{4.3}$$

Let $t_{ab} : N_a \to N_b$ be a tool bridge between the tools T_a and T_b. For a given version j of model $M_{a,j}$ we can calculate the corresponding model $M_{b,j}$ for tool T_b, as $M_{b,j} = t_{ab}(M_{a,j})$.

Diagram 4.4 provides a graphical representation of the relationship between models realizing different design decisions and models corresponding to different tool metamodels. On the left hand side is the refinement of model M_a using tool T_a, on the right hand side is the corresponding refinement of model M_b in tool T_b, where for any j the model $M_{a,j}$ and $M_{b,j}$ contain the same information. The concept is independent of the number of tools; in diagram 4.4 we exemplify the concept for two tools.

$$\begin{array}{ccc}
M_{a,0} & \xrightarrow{t_{ab}} & M_{b,0} \\
\downarrow{\scriptstyle d_{a,1}} & & \downarrow{\scriptstyle d_{b,1}} \\
\dots & \xrightarrow{t_{ab}} & \dots \\
\downarrow{\scriptstyle d_{a,k}} & & \downarrow{\scriptstyle d_{b,k}} \\
M_{a,k} & \xrightarrow{t_{ab}} & M_{b,k} \\
\downarrow{\scriptstyle d_{a,k+1}} & & \downarrow{\scriptstyle d_{b,k+1}} \\
\dots & \xrightarrow{t_{ab}} & \dots \\
\downarrow{\scriptstyle d_{a,j}} & & \downarrow{\scriptstyle d_{b,j}} \\
M_{a,j} & \xrightarrow{t_{ab}} & M_{b,j}
\end{array} \tag{4.4}$$

Diagram 4.4 visualizes the possible evolution paths of model $M_{a,0}$ into model $M_{b,j}$. In the following we describe one of the possible paths: An initial model $M_{a,0}$,

which might be empty, is developed in tool T_a. It is refined by making design decisions $d_{a,1}$, .., $d_{a,k}$, resulting in model $M_{a,k}$. To develop model $M_{a,k}$ further, tool T_b has to be used; thus model $M_{a,k}$ needs to evolve into model $M_{b,k}$. The tool bridge t_{ab} is used to evolve $M_{a,k}$ into $M_{b,k}$. Further design decisions $d_{b,k+1}$, .., $d_{b,j}$ are made in tool T_b, resulting in model $M_{b,j}$, the final model of development.

An example of a real-world evolution scenario matching the formal description above, is described in section 1.2.1. A model of a new automotive brake-by-wire system is developed in MATLAB/Simulink (c.f. figure 1.2). Several design decisions are made while the Simulink model is refined ($d_{a,1}$, .., $d_{a,k}$), among them is the decision to replicate a subsystem to increase its reliability. From the structure of the existing Simulink model either source code or a UML model might be generated. Tool integration technology (t_{ab}) translates the Simulink model into a corresponding UML model [23]. Afterwards the UML model is refined ($d_{b,k+1}$, .., $d_{b,j}$). In this scenario both horizontal and vertical model evolution occur.

In the following sections we describe work on supporting both horizontal and vertical model evolution. In section 4.2 we present work on horizontal model evolution, which was performed in the context of tool integration for automotive embedded systems development with the architecture description language EAST-ADL2. Horizontal model evolution changes the abstract or concrete syntax of a model, but keeps the information contained in the model unchanged. In section 4.3 we present work on vertical model evolution, which was done in the context of design decision documentation. Vertical model evolution changes the information contained in a model, but keeps the abstract and concrete syntax of the information unchanged.

In this work both the horizontal and the vertical dimension of model evolution are described by model transformations. The dimension of horizontal model evolution is realized by tool bridges using exogenous model transformations. The dimension of vertical model evolution is realized by executable design decisions using endogenous model transformations.

4.2 Horizontal Model Evolution: Tool Integration in Automotive Embedded Systems Development

This section addresses RQ 1 : *How can development tools used for the development of automotive embedded systems be integrated and exchange tool data?*

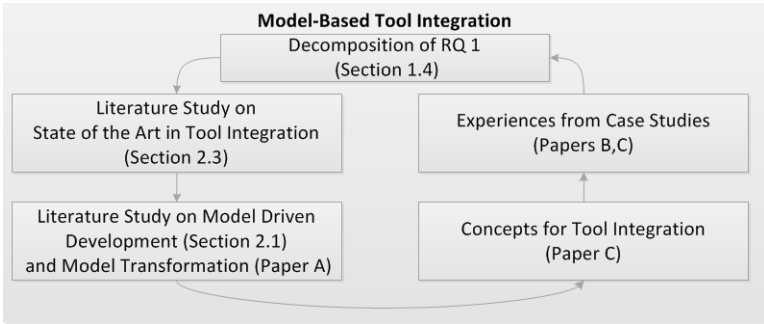

Figure 4.2: Approach for RQ 1

We use model-based tool integration mechanisms in order to realize support for early verification and analysis in model-driven development of automotive embedded systems with EAST-ADL2. In the following we focus on data integration, which is one of the five dimensions of tool integration [219].

To realize data integration in our case, a *design-oriented* EAST-ADL2 model has to evolve into an *analysis-oriented* model, involving a change of the metamodel. This allows the analysis of EAST-ADL2 models without requiring extensive knowledge of analysis methodologies, languages and tools. As a result, the tools for design and for analysis are integrated into a partial tool chain.

4.2.1 Requirements and Analysis

In this section, we first describe the requirements for data integration and introduce an architecture for data integration that is based on model transformation. We then describe how we use this architecture to realize partial tool chains for the early analysis of EAST-ADL2 models using the tool HiP-HOPS for safety analysis and the tool MATLAB/Simulink for simulation.

Desirable properties and requirements of data integration are:

- Automation of the integration to improve the efficiency and correctness of the data integration.
- Transparency of the integration to the user.
- Evolvability of the integration to deal with the evolution of the integrated tools.

We work under the assumption that the tools themselves cannot be changed to accomplish the data integration. Instead we use an adapter to make the data of tool B available to tool A. The adapter expresses the data of tool B in such a way that it can be used by tool A. In this case, tool A is the target tool and tool B is the source tool. In some cases, the bidirectional exchange of tool data is of interest, where both tools need to work with the tool data of the respective other tool.

During data integration some properties of the data need to be changed, while other properties need to be preserved. The following properties of the data might need to be *changed* by data integration:

- *Change of the abstract syntax:* The structure or syntax of the data might need to be adapted to the target tool. The information of the source tool needs to be rearranged and reinterpreted to adhere to the structure expected by the target tool. This step is independent of the technical space used by the target tool.

- *Change of the technical space:* Models can be represented using different technical spaces, such as XML, EMF or text-based grammars. The information needs to be represented by the technical space expected by the target tool.

The above mentioned properties are orthogonal to each other and can be changed independently. It is for example possible to change the technical space, while preserving the abstract syntax of a model.

The following properties of the data might need to be *preserved* for data integration:

- *Preservation of the graphical layout*: Models are often presented to the user in a graphical form. The user can convey information by the graphical placement of model elements. For this reason the graphical layout should be preserved by data integration.

- *Preservation of the semantics*: To preserve the semantics of a model in tool integration, there needs to be an overlap of the semantics between source and target metamodel. A data integration is semantics preserving if the meaning of source and target models is the same, even though it is represented in a different technical space or using a different abstract syntax. If the integrated tools are similar, the overlap in semantics of the metamodels is usually given and a semantics preserving data integration can be found. If the integrated tools are so different that there is little overlap in semantics between source and target metamodel, the mapping between the tools might only *approximate* a preservation of the meaning of the model. If the integrated tools are fundamentally different, there is no overlap in semantics and a preservation of the semantics might be impossible.

4.2.2 Solution

We use technology of model-based tool integration to realize data integration. We describe tool data with metamodels and express the correspondence between tool data of different tools in the form of model transformations. In our solution we use the principle of separation of concerns to structure the data integration in such a way that the independence of the two issues for changing the data is ensured. We propose an approach for decomposing the data integration into two bridges, connected by an intermediate model (see figure 4.3).

- The *technical space for integration* is chosen as a platform for the integration. The technical space for integration is independent of the technical space of any particular tool.
- The *intermediate model* is the tool-specific model that representS tool data in the technical space for integration. For each integrated tool we define an intermediate model. The structure of the intermediate model is similar to the structure of the tool data. Intermediate models correspond to intermediate metamodels.
- The *technical space bridge* changes the technical space. It has the purpose of translating data between the technical space of the tool and the technical space of the intermediate model. Technical space bridges can be realized for example by model-to-text transformations and text-to-model transformations (parsers); a list of the different mechanisms is provided in paper C. Since the intermediate model is defined to have a structure similar to the tool data, technical space bridges preserve the structure of the tool data.
- The *structural bridge* maps corresponding elements of the intermediate model of one tool to the intermediate model of another tool in a semantics-preserving way. To establish the semantics-preserving correspondence, an understanding of the semantics of both intermediate metamodels and their corresponding tools is required. Model-to-model transformations are used as a mechanism for realizing structural bridges. For structural bridges we assume:
  - the semantics of source and target metamodel overlap, so a semantics-preserving mapping can be found.
  - relevant tool data is available as an intermediate model in a common technical space for integration.

In figure 4.3 we visualize how we use these concepts to realize a partial tool chain involving the simulation tool MATLAB/Simulink, the systems engineering language EAST-ADL2 and the safety analysis tool HiP-HOPS. In this tool chain, the data of Simulink is first translated into an *intermediate model* using a *technical space bridge*, then a *structural bridge* maps concepts to EAST-ADL2. No technical space bridge is required for EAST-ADL2, as the data of EAST-ADL2 is natively

represented in the technical space of integration. To integrate EAST-ADL2 with HiP-HOPS, a structural bridge maps the data of EAST-ADL2 to the intermediate model of HiP-HOPS. A technical space bridge maps the concept to the tool-specific format of HiP-HOPS.

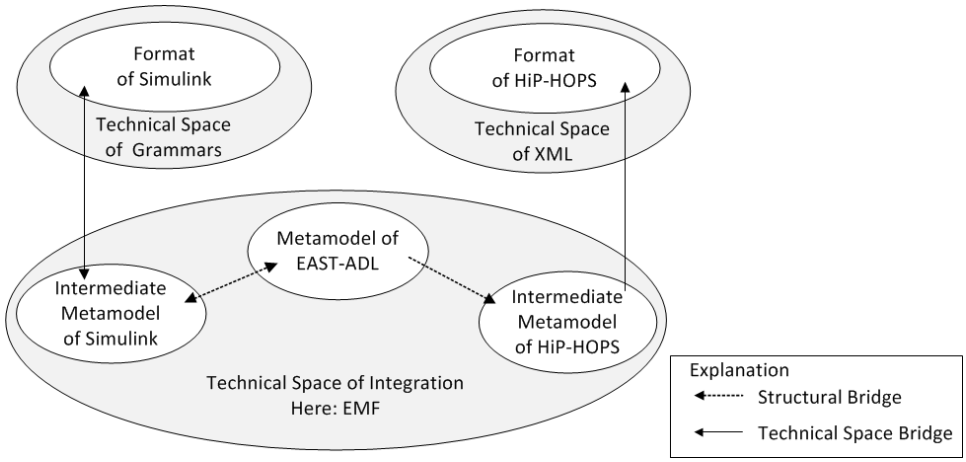

Figure 4.3: Technical Space Bridges and Structural Bridges

To implement the bridges, we select model transformation technology that suits each of the bridges individually. A large number of model transformation languages, engines and tools are available, which have different characteristics that can be relevant for certain transformation tasks. An overview and a classification of model transformation technology is presented in the appended paper A. Important factors in the selection of a model transformation tool are the support of the technical spaces involved and the properties of the transformation tool.

To validate the presented tool integration approach, we perform two case studies that connect EAST-ADL2 to MATLAB/Simulink and to the safety and reliability analysis tool HiP-HOPS. We report on the common experiences from these integration approaches in appended paper C.

The automotive industry has a growing demand for the seamless integration of safety analysis tools into the model-based development tool chain for embedded systems. This requires translating concepts of the automotive domain to the safety domain. We automate such a translation between the automotive architecture description language EAST-ADL2 and the safety analysis tool HiP-HOPS by using model transformations and by leveraging the advantages of different model transformation techniques. By means of this integration, safety analysis can be conducted early in the development process, when the system can be redesigned to fulfill safety goals with relatively low effort and cost. We study the tool integration approach between EAST-ADL2 and HiP-HOPS in depth in appended paper B.

The latest version of the integration between HiP-HOPS and EAST-ADL2 considers the crosscutting nature of safety in embedded systems (see section 1.1.1). Embedded systems consist of both hardware and software components and a description of the allocation of software components to hardware components. The system as a whole needs to be safe and reliable, so both software and hardware aspects need to be considered by a safety analysis. Since safety is a crosscutting system property, it is not sufficient to analyze the safety for software and hardware components separately. On the contrary, we need to consider the interaction and relationship between software and hardware components for a safety analysis of the system. This comprehensive view is relevant for the accuracy of safety analysis, as the software may be able to detect and mitigate hardware problems.

We use the architecture description of EAST-ADL2, which describes software components, hardware components and the allocation of software to hardware components. We export this crosscutting view of the system including software and hardware to HiP-HOPS using structural and technical space bridges. As a result, the safety analysis tool HiP-HOPS can analyze the system as a whole, considering software aspects, hardware aspects and their interaction.

4.2.3 Related Work

Data integration in the context of model-based tool integration can be realized by a number of mechanisms, as introduced in section 2.3.1. When model transformations are used as a mechanism for data integration, a usual assumption is that both source and target models are represented in the same technical space. Thus, many approaches focus only on the structural bridge [72, 130, 60].

The MOFLON tool integration approach [4], however, is similar to our approach, as it differentiates between tool adapters, which correspond to technical space bridges, and transformation rules, which are comparable to structural bridges. The MOFLON approach uses model transformations with triple graph grammars and MOF as the technical space for integration.

A tool integration approach for the automotive domain with a similar architecture integrates SysML and AUTOSAR using bidirectional model synchronization [91]. It translates between two design models of different level of abstraction to support the synthesis process. Thus, it has a different goal than our approach. Our approach translates between a design model in EAST-ADL2 and analysis models in Simulink and HiP-HOPS to allow early verification through simulation and safety analysis.

Our approach defines mappings for data integration on metamodel (M2) level. Approaches for metametamodel (M3) level bridges provide a high degree of automation, as both metamodels and transformation rules for structural bridges can be generated automatically from a mapping of concepts on metametamodel level [126, 125, 124, 36]. However, to achieve the high level of automation, some assumptions have to be made: (1) The target metamodel cannot be user-defined, but needs to be generated automatically as part of the approach. (2) This approach

only considers homogeneous tools from the same domain with only structural differences and cannot be used for heterogeneous tools where semantics-preserving mappings need to be specified based on domain knowledge. (3) The approach does not consider technical space bridges. These three assumptions are not fulfilled in the case of the data integration of EAST-ADL2, HiP-HOPS and Simulink.

4.3 Vertical Model Evolution: Design Decision Documentation for Models

This section addresses RQ 2 : *How can design decisions be documented in model-driven development?*

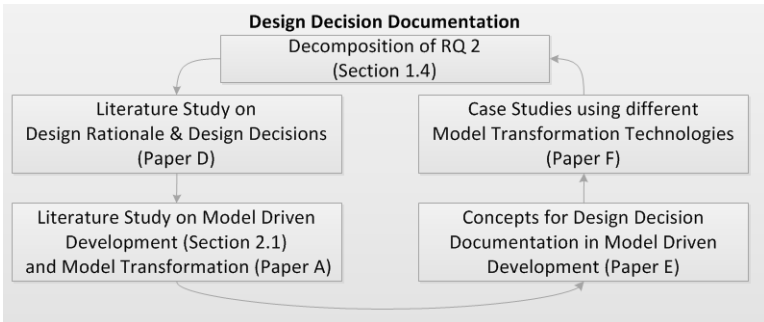

Figure 4.4: Approach for RQ 2

Models are developed by making a number of design decisions and subsequently updating the model to reflect the design decisions. The effect, which the design decision has on the model, is captured by the model. However, the knowledge about the design decision itself, such as the fact that a design decision was made, why it was made and what the decision was, is usually not made explicit. Thus, this knowledge is easily lost, a phenomenon known as *knowledge vaporization* [29]. However, this knowledge is often critical for continued development and maintenance. As a countermeasure, design decisions can be documented explicitly. The main inhibitors are the overhead induced by capturing design decisions and the lack of proper tool support [198, 197]. Model-driven development follows the paradigm of using models to represent, store, analyze and simulate architectural knowledge. However, design decisions do not have an explicit representation in model-driven development, yet.

4.3.1 Requirements and Analysis

Desirable properties and requirements of design decision documentation for models are:

- Explicit documentation of the design decision, for example by a textual description.
- Applicability for an arbitrary metamodel: it should not be necessary to change existing metamodels to document design decisions.
- Low capture overhead: the effort for linking the documentation to the model should be low.
- Link between the design decision documentation and the model: a link between the design decision documentation, the context of the design decision and the model elements affected by the design decision.
- Consistency between the design decision documentation and the model: the change described by the design decision documentation should be consistently realized in the model.

Oliver [164] stresses the importance of executable models, i.e. models that can be interpreted by engineers as well as by computers. It is the basis for automating engineering activities and can support the decision making of the engineer, both in terms of quality and efficiency. We thus propose an executable representation of design decisions in this work, called *Executable Design Decisions*. To achieve executability of a design decision representation, a couple of things have to be in place: The design decisions need to be described with sufficient detail, so they can be executed. An interpreter needs to be available that can execute the description of design decisions and can produce a modified architecture as output. Using a generative approach, the representation of design decisions has a double function: it can be used both for synthesis and for documentation of the model.

4.3.2 Solution

We propose to use model transformations as an executable representation of design decisions. Model transformations can be executed, the result of the execution is the changed architectural model. This has the advantage that design decisions only need to be captured in the form of a model transformation and the change in the architecture is merely a cause of the execution of the transformation. Model transformations can be used for models conforming to different metamodels. Thus, this approach is not bound to a specific notation. It can be applied to architectural models that are described using existing metamodels or architecture description languages. Model transformations provide a *mechanism* for representing important parts of the design decisions. Figure 4.5 illustrates the correspondence between the concepts of model transformation and the concepts of design decisions.

In order to identify to what extent design decisions can be represented by model transformations, we examine which information is required for the representation of design decisions. As an orientation, we use the ontology of architectural design decisions by Kruchten [133]. According to this ontology, the description of a design

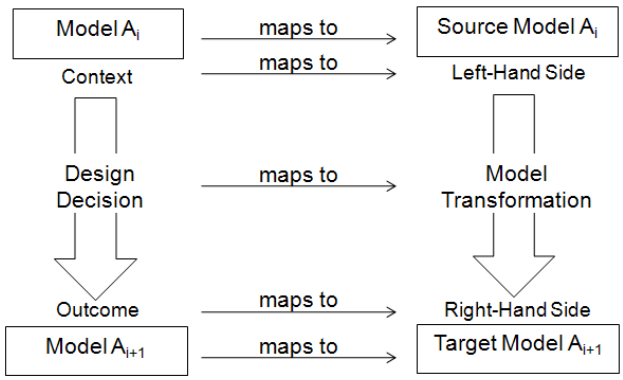

Figure 4.5: Model transformation as a mechanism for representing parts of a design decision

decision comprises a description of the decision, the scope of the decision, the rationale of the decision, cost, risk, author, timestamp and state. We found that the elements of the design decision ontology can be partly represented by model transformations:

- The scope of the decision describes the context, in which the decision is defined. In the ontology, Kruchten lists different interpretations of the scope, such as system scope, time scope and organization scope. We focus here on the system scope. The context is the precondition under which the design decision can be applied and executed. The context corresponds to the matching patterns or the left-hand side of the model transformation rule.

- The outcome of the decision can be expressed by the right hand-side of a model transformation rule. The outcome describes the added, changed or deleted model elements. The outcome of a decision can also be regarded as the postcondition of the decision.

In summary, the core of the design decision, consisting of the decision itself and its application context, can be described by model transformations. The model transformation as a representation of design decisions fulfills two functions simultaneously: documenting the design decision and changing the architecture by executing the transformation. The design decision representation is compatible with existing metamodels, as long as they can be processed by model transformation tools.

The additional elements mentioned in the ontology of design decisions, can be regarded as an extension to the core of the decision. We store them textually in a separate data structure called *Design Decision Management Container (DDMC)*. This container is used for managing the metadata of each design decision in textual form, such as rationale, cost, risk, author, timestamp and state. Furthermore, the DDMC points to the model transformation realizing the decision.

In the following we present some of the properties of the proposed executable representation of design decisions:

- We *explicitly* represent technical design decisions that have an effect on the model.
- Design decisions are represented independently of a certain modeling language. Model transformations can be specified for any metamodel without needing to change the metamodel.
- In stepwise model refinement the capturing of design decisions comprises four steps:

  1. Performing an update of the model, so it reflects the new design decision
  2. Documenting *where* in the model the change applies and *what* is changed
  3. Documenting the rationale of the change
  4. Linking the documentation to the model elements affected by the change

  The overhead of these steps is one of the main reasons why design decisions are not documented [134]. Tools can help to *reduce the overhead* for the modeler, by exploiting the intrinsic relation between steps (1), (2) and (4). A model transformation has a precondition and a postcondition that can be used for documenting *where* in the model the change applies and *what* is changed (2). When the model transformation is executed on the initial model, the outcome is the updated model (1). The precondition of the model transformation links the design decision documentation to the model before the change and the postcondition links the design decision documentation to the model after the change (4); this link can even be automatically documented by the transformation engine in the form of trace links. In summary, a model transformation is capable of capturing steps (1), (2) and (4) by a single representation and only step (3) needs to be captured separately.
- The documentation of design decisions is linked to the model elements affected by the decision. The core of the executable design decision is a model transformation, which describes both the *context* in which the design decision can be made and the *effect* that the design decision will have on the model. The executable design decision is thus linked to the model elements that are affected by the decision.
- Document-based approaches separate documentation from the system model. Due to this separation, the model and the documentation can evolve separately. Over time, the documentation is prone to become inconsistent with the model and may contain contradictions, inconsistency, incompleteness, redundancy and incorrectness. In document-based approaches, information is not traceable, so the origin and effect of the information cannot be easily

retrieved. On the contrary, the proposed documentation of design decisions ensures *consistency* with the architectural model. Since we use model transformations to represent design decisions, we can execute the design decision representation and inspect the effect the design decision has on the model. As a consequence of this automation, the model and the design decision representation are automatically synchronized and consistent.

We evaluate the approach by a case study in the context of models for automotive embedded systems. We demonstrate that it is feasible to express design decisions using different model transformation formalisms. The design decision can be described using endogenous, in-place model-to-model transformations. We used both ATL [116] and Tiger EMF [25] in the case study.

4.3.3 Related Work

A number of different approaches for representing design decisions have been proposed by academia. They can be split into informal representations, template-based approaches and structural approaches, as introduced in section 2.4. These approaches are targeted towards general software development and can be applied for model-driven development as well. However, they do not take the properties of model-driven development into account and do not use the possibilities of model-driven development, such as the enforcement of consistency and possibilities for automation. Thus, these approaches do not fulfill all requirements for design decision documentation for models put forth in the previous section:

- Explicit documentation of the design decision: The approaches for design decision documentation introduced in section 2.4 fulfill this requirement.
- Applicability for an arbitrary metamodel: The Archium [115] approach associates design decisions with modifications of the architecture. However, it requires that the Archium metamodel is used. Navarro et al. propose the ATRIUM metamodel for model-driven development [159]. The important metaclasses have text attributes for design decision and design rationale. Specific metamodels can provide the possibility for specifying design rationale together with the model. However, the approach requires a change in the metamodel, which might not be possible, when a given metamodel or language needs to be used.
- Low capture overhead: AREL is a UML profile that can be used to annotate architectural models with design decisions and rationale [199]. However, the links between models and design decision documentation need to be established and maintained manually. Thus, the capture overhead is relatively high.
- Link between the design decision documentation and the model: Approaches for representing design decisions that were developed in recent years focus on

linking design decisions with the architectural model. Older approaches, such as IBIS, PHI and QOC do not link decisions with affected model elements.

- Consistency between the design decision documentation and the model: The SEURAT approach links rationale to specific lines in the source code [37]. If source code lines are added, the link to the source code is inconsistent. Comments in UML diagrams do not enforce the consistency between the change in a model and a design decision documented in the UML comment [170].

4.4 Chapter Summary

In this chapter we presented the approach and results of this work. We identified two dimensions of model evolution that handle different types of changes in models:

- Horizontal model evolution: We presented an approach for data integration, using a common metamodel to connect specialized tools and their models. We decomposed data integration into modular bridges for adapting both the structure and the technical space. We applied this approach to build a partial tool chain for automotive embedded systems development involving a systems engineering tool, a safety analysis tool and a simulation tool.

- Vertical model evolution: We proposed a representation for design decision documentation in model-driven development. We first identified a number of requirements for the representation. We then showed to which extent these requirements are met by our proposed *executable design decision* representation for models and compared it to related work.

Chapter 5

Discussion

In this chapter we discuss the limitations, open issues and additional possibilities for application of the concepts in a different context.

5.1 Horizontal Model Evolution

Graphical Information

Models are often presented to the user in a graphical form. In graphical modeling, meaning can be conveyed by the graphical placement of model elements. The graphical layout can convey that certain elements belong together by placing them close to each other on the drawing canvas. When translating the model to another formalism or tool, this layout information might need to be preserved. One possible thread of future work could be the consideration of layout information for data integration of models.

Combination with other Tool Integration Technology

In section 2.3 we introduced a number of approaches for tool integration, among them the tool integration platforms ModelBus and Jazz. These approaches provide a repository with access control and mechanisms for control integration, but are not concerned with the tool-specific mapping of tool data. Our approach is focused on defining metamodels and data integration through model transformation, so it could be synergistic to combine both approaches.

Common metamodels for data integration are defined both in a domain-specific way, such as EAST-ADL or the CESAR Common Metamodel and in a domain-independent way, for example by the OSLC initiative or the STEP standard. It might be useful to compare the domain-specific metamodels EAST-ADL and the CESAR Common Metamodel with the generic resource definitions of OSLC [223].

Choice of Model Transformation Technology

If two tools in a tool integration solution need to access data of the respective other tool, bidirectional model transformations can be used for realizing the structural bridge. Bidirectional model transformation languages allow the expression of mappings between two metamodels that can be executed in both directions, from A to B and from B to A. As only one model transformation description is sufficient to express the mapping of both execution directions, the mapping between the languages could be expressed more concisely by a bidirectional transformation. The integration of MATLAB/Simulink and EAST-ADL2 is bidirectional, thus a bidirectional transformation language could be used. However, for technical reasons a bidirectional model transformation language could not be used, since the tools available at that time did not natively support the UML profile of EAST-ADL2. For the integration between HiP-HOPS and EAST-ADL2, only a unidirectional mapping is required.

Modularization of Model Transformation by Separation of Concerns

In paper A we show that the model transformation language needs to be appropriate to the problem we intend to solve by means of model transformation technology. In paper C we show how a complex tool integration problem can be solved by decomposing it into several simpler problems first. We use the principle of separation of concerns for decomposing the transformation and then choose an appropriate technology for each partial problem. This modularization approach may also be applicable to reduce the complexity of other model transformation problems. The approach can be summarized as follows:

- Identification of the different concerns handled by the transformation.
- Decomposition of the model transformation problem into a set of partial problems using the principle of separation of concerns.
- Selection of appropriate model transformation languages and technologies for each partial problem.
- Definition of an intermediate metamodel that serves as an interface between the transformations of partial problems.
- Usage of *external composition* [56], which is used to combine the partial transformations to a transformation chain.

The decomposition allows both using the advantages of different model transformation technologies and prepares the solution for evolution. The identified concerns may evolve separately without affecting each other, so only a partial transformation is affected by changes. The decomposition supports evolvability and the use of appropriate technology.

Model Transformation Co-Evolution

The definition of metamodels is central to model-driven development. The models and the model transformations are dependent on the definition of the metamodels and need to be adapted if the metamodel changes. The co-evolution of models and metamodels has been studied widely and solutions exist for adapting models automatically to cope with metamodel changes [102, 100, 101, 218, 158].

In a data integration solution, not only the model, but also the model transformations used for data integration need to co-evolve with the metamodel. The proposed model transformation solution with separation of concerns simplifies the co-evolution with the metamodel, as concerns can evolve separately without necessarily affecting other concerns.

From our experiences, the decomposition of the model transformation supports the co-evolution of the model transformation with the metamodel, when the metamodel changes frequently. The model transformation between EAST-ADL2 and HiP-HOPS, described in the appended papers B and C, needed to co-evolve with several changes in both the EAST-ADL2 and HiP-HOPS metamodels. After an initial tool integration was built, the HiP-HOPS language changed from a text-based representation to an XML representation, so the data integration solution needed to be adapted. Due to the modular organization of our model transformation, the necessary changes affected only the technical space bridge, leaving the structural bridge unaffected. Thus, the decomposition supported the co-evolution of model transformations.

Practical Evaluation

The tool integration approach has been evaluated by creating tool support for the integration of three different development tools: MATLAB/Simulink, HiP-HOPS, PapyrusUML with the EAST-ADL profile. We have evaluated the tool support for integration with small case studies. This can be complimented with an additional evaluation using industrial sized case studies.

5.2 Vertical Model Evolution

Limitations

The classification scheme for design rationale described in paper D differentiates between technical and non-technical rationale and between the different levels of justification (R0 - Artifact Level, R1 - Decision Level, R2 - Decision Rationale Level, R3 - Rationale Rationale Level). In this work, we focus on the first level of justification provided by design decisions, also called the R1-level. Rationale on R2 and R3 level include decision rationale and trade-off decisions and are not considered in this work. We also focus on technical design decisions that can be linked to the

architectural model. Non-technical design decisions, such as managerial decisions about staffing or project management are not considered in this work.

Comparison to Version Management

There are three basic approaches for capturing model evolution: state-based, change-based and operation-based approaches [103].

- State-based approaches save snapshots of the model as versions or states. They are used widely today in the form of version management and software configuration management systems, such as CVS or SVN. A survey on model versioning [3] provides a list of criteria and a comparison of state of the art state-based approaches for model versioning.
- Change-based approaches represent the differences between two models.
- Operation-based approaches record the operations that transfer one version of a model into another one.

These approaches can be differentiated by the precision of the tracked changes. State-based approaches may use data mining techniques to extract trace links from the history of the versioning system [58]. Since the used data mining approaches for extracting the traces are heuristics, they can only provide an approximation. Operation based approaches can track the changes in models with a higher precision than the other approaches [103], due to precise trace links. The documentation of design decisions with our approach for *Executable Design Decisions* is an operation-based approach, as it is based on capturing change by operations in the form of model transformations. The traces generated by a model transformation are a more reliable source of information than the traces extracted by heuristics.

Practical Evaluation

In our case studies we have shown that it is feasible to represent design decisions using model-transformations and that the approach is usable. The next step is an evaluation of the usefulness of this representation. More descriptive work can be done to practically evaluate the proposed methods and tools.

- A larger academic or industrial case study can be performed to learn about potential challenges in the practical application.
- An industrial case study can show if design decision documentation using model transformations can be used in current industrial practices or which changes to current practices would be necessary.
- An empirical study can examine if design decision documentation actually improves maintenance tasks and by how much. Other empirical studies on design decision documentation support these claims [77, 78]. Bratthall et al.

[31] studied the impact of design rationale on maintenance work in an empirical study. When design rationale was present, the time spent on maintenance tasks was reduced and the correctness was improved compared to the case without design decision documentation. Another independently performed empirical study [121] supports these findings.

5.3 General

Model-Driven Development

Model-driven and model-based approaches promise to manage the complexity inherent in the development of embedded systems, to improve efficiency, improve quality and to provide an early analysis and simulation of various system properties [69]. However, there is still no generalizable evidence that model-driven development fulfills the promises it makes in large-scale industrial applications [155]. This thesis does not provide generalizable evidence, but shows possibilities how model-driven development can contribute to issues that are insufficiently addressed in general, such as tool integration and the consistent documentation of design decisions. The technology and methodology of model-driven development enable the proposed solutions.

Scope and Applicability

The work on tool integration and design decision documentation has been performed in the context of model-driven development of automotive embedded systems. The developed methods and concepts are independent of the automotive domain. The proposed methods and concepts can be transferred to other domains that use model-driven development.

Chapter 6

Future Work

In this chapter we provide a plan for future work, including a list of research questions guiding the work. Figure 6.1 visualizes the placement of the future work in the overlap of the areas of relevance. In future work we would like to address:

- Documenting design decisions in multi-tool environments.
- Reusing design decisions as design decision patterns.
- Reconstructing design decisions from model differences.
- Supporting management and evaluation of design decisions.

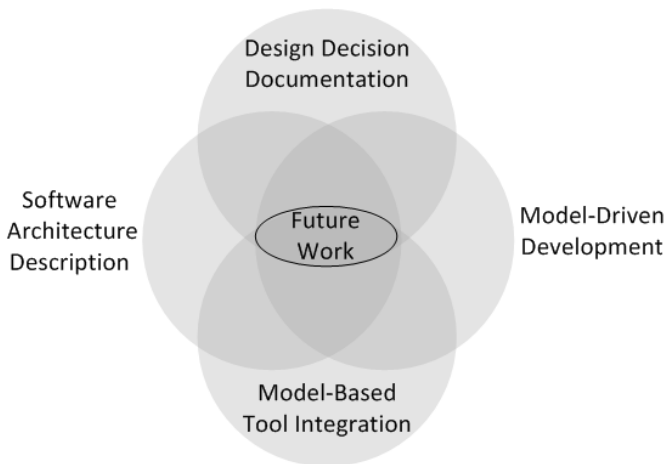

Figure 6.1: Future work on the thematic map

6.1 Design Decision Documentation in Multi-Tool Environments

With the current approach of adopting model-based tool integration technology, the focus is on exchanging models between tools. In this process, important meta-data of the models, such as rationale and design decisions, may be lost. This makes the models hard to understand and may lead to model inconsistencies, model erosion and maintenance problems. We propose an integrated approach for documenting design decisions during model refinement and automatically transferring not only models but also the associated design decision documentation between different tools. As a result, design decision documentation can be linked consistently to corresponding model elements of the various tool-specific models, easing model evolution across several development tools.

So far, the documentation of design decisions is not a major concern in tool chain approaches. Model-based tool integration approaches focus on translating tool-specific models, not the associated design decisions. If a design decision documentation exists for a tool-specific model A, and the model is translated into another tool-specific model B, the design decision documentation of A is not automatically translated and linked to model B. Documented design decisions in model A are lost when continuing development with the automatically translated model B.

We would like to find out if it is possible to transfer a design decision documentation from tool A to tool B in such a way that the design decision documentation will be linked to corresponding model elements in both tools. We would also like to automate this transfer of the design decision documentation. We propose to combine ideas from design decision documentation and from model-based tool integration.

6.2 Design Decision Patterns

Our representation of design decisions explicitly documents the context and the outcome of the design decision, where the context is specific to one particular location in the associated model. For a given executable design decision we can modify the description of the context to make the design decision applicable in another context. By widening the context and introducing parameters into the context description, the design decision becomes a design decision pattern.

Using the representation of executable design decisions, design decision patterns can be represented as model transformations with parameters and organized in a catalog of design decision patterns. The patterns of this catalog can potentially be reused in different projects. Design decision patterns also provide a way to formally represent design pattern [87], which are typically described textually.

6.3 Design Decision Reconstruction from Model Differences

New design decisions can be entered into the system in three ways: (1) creating a new executable design decisions manually, this involves writing a model transformation to describe the change, (2) reusing an existing executable design decision by instantiating a design decision pattern and binding its parameters to specific values, (3) calculating an executable design decision from the model difference to a previous version.

The last options allows designers to work with the model as usual and retrospectively extract the executable design decision from the model difference. This retrospective analysis of design decisions can also be used for extracting design decisions from a version management system. This can be used for assessing commonly used design decisions in embedded systems and for eliciting decision patterns.

6.4 Design Decision Management

The design decision documentation needs to be stored, archived and managed, especially for large projects with many design decisions. To achieve this, our representation for a design decision documentation can be embedded into an information structure for managing several design decisions and their relations. Such a management infrastructure can also represent additional information, such as design alternatives and trade-offs.

We envision a decision management system similar to CVS that stores sequences of executable design decisions. It allows storing several design decisions as sequences of executable design decisions. Thus, the model corresponding to the design decisions up to any chosen point in the sequence can be calculated. In addition, it is possible to add, change or remove a design decision at an arbitrary point in the sequence. The subsequent design decisions in the sequence can take this change into account. This can be used as a tool for change impact analysis, since the impact of a modified or newly introduced design decision on the system and on other design decisions can be assessed.

The management infrastructure also allows expressing alternative options for each design decision. A design decision can be evaluated and compared against its alternatives by automatically analyzing the model produced by it. The effect a design decision has on certain system property can be evaluated immediately using the analysis described in section 4.2. This allows making an analysis-based assessment of the design decisions using *what-if-scenarios*.

6.5 Research Questions

The research questions for guiding the future work are: How can design decision documentation be managed and used in scenarios with multiple integrated tools?

1. How can a model-based tool integration approach be extended to exchange the design decision documentation in addition to the tool-specific model? In which engineering scenarios does the transfer of design decision documentation make sense?

2. How can design decisions be captured and stored as reusable design decision patterns?

3. How can executable design decisions be reconstructed from model differences or from a model version history?

4. How can design decisions be managed? How can we evaluate and compare alternative design decisions using automated analysis of system properties?

Chapter 7

Concluding Remarks

This thesis is an intermediate milestone, many open issues and areas of future work exist. In this chapter we summarize the main contributions of this work.

This thesis has the goal to support model evolution in model-driven development of automotive embedded systems. To support model evolution, we first identify two dimensions of model evolution that handle different types of changes in models, as illustrated in diagram 7.1.

- A model $M_{a,0}$ can evolve by changing the metamodel it conforms to, resulting in $M_{b,0}$. This is depicted by the horizontal arrow t_{ab} in equation 4.1. This type of evolution is called *horizontal model evolution*.
- A model $M_{a,0}$ can evolve by adding, deleting or changing model elements, resulting in $M_{a,1}$. This is depicted by the vertical arrow $d_{a,1}$ in equation 4.1. This type of evolution is called *vertical model evolution*.

$$\begin{array}{ccc} M_{a,0} & \xrightarrow{t_{ab}} & M_{b,0} \\ \Big\downarrow{\scriptstyle d_{a,1}} & & \\ M_{a,1} & & \end{array} \tag{7.1}$$

In this thesis we propose support for both horizontal and vertical model evolution:

- *RQ 1: How can development tools used for the development of automotive embedded systems be integrated and exchange tool data?*
  This question addresses *horizontal model evolution*, where models evolve by being translated. Models are adapted to conform to other metamodels and thus become accessible in other tools. We present an approach for tool integration that focuses on the dimension of data integration. The approach uses a common metamodel to connect specialized tools and their models. We

identify the separate concerns addressed by a data integration approach and use the principle of separation of concerns to decompose the data integration into separate modules: the technical space bridge and the structural bridge. For each bridge we select an appropriate model transformation technology. The basis for this selection is a systematic survey of model transformation technologies. We apply this approach in a case study and integrate several tools for automotive embedded systems development: A systems engineering tool, a safety engineering tool and a simulation tool.

- *RQ 2: How can design decisions be documented in model-driven development?* This question addresses *vertical model evolution*, where models evolve by adding, deleting or changing model elements. We systematically survey approaches for design rationale and design decision documentation as a means for manging and documenting vertical evolution. We propose a design decision representation that is targeted to model-driven development. This design decision representation links documentation and model, lowers capture overhead through partial automation, ensures consistency between model and documentation through automation and is independent of a particular metamodel. We show in a case study how this representation can be used to document the stepwise refinement of models.

For future work, we plan to combine approaches for horizontal and vertical model evolution to support model evolution in multi-tool environments. We extend the tool integration approaches (horizontal dimension) to not only transfer models between different tools, but to also transfer design decision documentation (vertical dimension). Consistent design decision documentation can thus be made available across corresponding models in the tool chain.

Bibliography

[1] Adamsson, Niklas, "Model-based Development of Mechatronic Systems - Reducing the Gaps between the Competencies?" in *Proc. of the TCME*, April 2004.

[2] Adler, Terry, Black, Janice A., and Loveland, John P., "Complex Systems: Boundary-Spanning Training Techniques," *Journal of European Industrial Training*, vol. 27, pp. 111–124, 2004.

[3] Altmanninger, Kerstin, Seidl, Martina, and Wimmer, Manuel, "A Survey on Model Versioning Approaches," Johannes Kepler University Linz, Tech. Rep., 2009. [Online]. Available: http://smover.tk.uni-linz.ac.at/docs/IJWIS09_paper_Altmanninger.pdf

[4] Amelunxen, C., Klar, F., A. Königs, T. Rötschke, and A. Schürr, "Metamodel-based tool integration with MOFLON," in *ICSE '08*, 2008, pp. 807–810. [Online]. Available: http://dx.doi.org/10.1145/1368088.1368206

[5] ARTEMIS Industry Association, "ARTEMIS strategic research agenda," ARTEMIS, Tech. Rep., 2006. [Online]. Available: https://www.artemisia-association.org/downloads/SRA_MARS_2006.pdf

[6] ——, "ARTEMIS strategic research agenda - design methods and tools," ARTEMIS, Tech. Rep., 2006. [Online]. Available: https://www.artemisia-association.org/downloads/RAPPORT_DMT.pdf

[7] Augsburg, Tanya, *Becoming Interdisciplinary: An Introduction to Interdisciplinary Studies*. Kendall/Hunt Publishing Company, January 2005. [Online]. Available: http://www.worldcat.org/isbn/0757515614

[8] AUTOSAR Consortium. Automotive open software architecture (AUTOSAR). [Online]. Available: http://autosar.org/

[9] Babar, Muhammad A., Northway, Andrew, Gorton, Ian, Heuer, Paul, and Nguyen, Thong, "Introducing Tool Support for Managing Architectural Knowledge: An Experience Report," in *IEEE International Conference on the Engineering of Computer-Based Systems*, vol. 1. Los Alamitos, CA, USA: IEEE, 2008, pp. 105–113. [Online]. Available: http://dx.doi.org/10.1109/ECBS.2008.27

[10] Babich, Wayne A., *Software Configuration Management: Coordination for Team Productivity*. Addison Wesley Longman, February 1986. [Online]. Available: http://www.worldcat.org/isbn/0201101610

[11] Baker, Paul, Loh, Shiou, and Weil, Frank, "Model-Driven Engineering in a Large Industrial Context - Motorola Case Study," in *Model Driven Engineering Languages and Systems*, ser. Lecture Notes in Computer Science, Briand, Lionel and Williams, Clay, Eds., vol. 3713. Berlin, Heidelberg: Springer Berlin / Heidelberg, 2005, pp. 476–491. [Online]. Available: http://dx.doi.org/10.1007/11557432_36

[12] Basili, Victor R., "The role of experimentation in software engineering: past, current, and future," in *ICSE '96: Proceedings of the 18th international conference on Software engineering*. Washington, DC, USA: IEEE Computer Society, 1996, pp. 442–449. [Online]. Available: http://portal.acm.org/citation.cfm?id=227818

[13] Baumgart, Andreas, "A common meta-model for the interoperation of tools with heterogeneous data models," in *3rd Workshop on Model-Driven Tool & Process Integration (MDTPI2010) at the European Conference on Modelling Foundations and Applications (ECMFA2010)*, June 2010.

[14] Bernstein, P., "Applying model management to classical meta data problems," 2003. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.12.772

[15] Bernstein, Philip A. and Melnik, Sergey, "Model management 2.0: manipulating richer mappings," in *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM, 2007, pp. 1–12. [Online]. Available: http://dx.doi.org/http://doi.acm.org/10.1145/1247480.1247482

[16] J. Bézivin, Bouzitouna, Salim, Del Fabro, Marcos, Gervais, Marie P., F. Jouault, Kolovos, Dimitrios, Kurtev, Ivan, and Paige, Richard F., "A Canonical Scheme for Model Composition," in *Model Driven Architecture Foundations and Applications*, 2006, pp. 346–360. [Online]. Available: http://dx.doi.org/10.1007/11787044_26

[17] J. Bézivin and O. Gerbé, "Towards a Precise Definition of the OMG/MDA Framework," in *ASE '01: Proceedings of the 16th IEEE international conference on Automated software engineering*. Washington, DC, USA: IEEE Computer Society, 2001, p. 273.

[18] Biehl, Matthias, "Documenting Stepwise Model Refinement using Executable Design Decisions," in *Proceedings of the International Workshop on Models and Evolution (ME 2010)*, October 2010.

[19] ——, "Literature Study on Design Rationale and Design Decision Documentation for Architecture Descriptions," Royal Institute of Technology, Tech. Rep. ISRN/KTH/MMK/R-10/06-SE, July 2010.

[20] ——, "Literature Study on Model Transformations," Royal Institute of Technology, Tech. Rep. ISRN/KTH/MMK/R-10/07-SE, July 2010.

[21] Biehl, Matthias, DeJiu, Chen, and M. Törngren, "Integrating Safety Analysis into the Model-based Development Toolchain of Automotive Embedded Systems," in *Proceedings of the ACM SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems (LCTES 2010)*, April 2010, pp. 125+.

[22] Biehl, Matthias and W. Löwe, "Automated Architecture Consistency Checking for Model Driven Software Development," in *Proceedings of the Fifth International Conference on the Quality of Software Architectures (QoSA 2009)*, June 2009, pp. 36–51. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-02351-4_3

[23] Biehl, Matthias, C.-J. Sjöstedt, and M. Törngren, "A Modular Tool Integration Approach - Experiences from two Case Studies," in *3rd Workshop on Model-Driven Tool & Process Integration (MDTPI 2010) at the European Conference on Modeling Foundations and Applications (ECMFA 2010)*, June 2010.

[24] Biehl, Matthias and M. Törngren, "An Executable Design Decision Representation using Model Transformations," in *36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 2010)*, September 2010.

[25] Biermann, E., Ehrig, K., C. Köhler, Kuhns, G., Taentzer, G., and Weiss, E., "Graphical definition of in-place transformations in the Eclipse Modeling Framework," in *MODELS 2006*, vol. 4199, 2006, pp. 425–439. [Online]. Available: http://dx.doi.org/10.1007/11880240_30

[26] BITKOM, "Studie zur Bedeutung des Sektors Embedded-Systeme in Deutschland," BITKOM Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e. V., Tech. Rep., November 2008. [Online]. Available: http://www.bitkom.org/files/documents/Studie_BITKOM_Embedded-Systeme_11_11_2008.pdf

[27] Blanc, Xavier, Mounier, Isabelle, Mougenot, Alix, and Mens, Tom, "Detecting model inconsistency through operation-based model construction," in *ICSE 2008*. New York, NY, USA: ACM, 2008, pp. 511–520. [Online]. Available: http://dx.doi.org/http://doi.acm.org/10.1145/1368088.1368158

[28] Blessing, Lucienne T. M. and Chakrabarti, Amaresh, *DRM, a Design Research Methodology*, 1st ed. Springer, June 2009. [Online]. Available: http://www.worldcat.org/isbn/1848825862

[29] Bosch, Jan, "Software Architecture: The Next Step," *Software Architecture*, vol. 3047, pp. 194–199–199, 2004. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-24769-2_14

[30] Bradley, D. A., "The what, why and how of mechatronics," *Engineering Science and Education Journal*, vol. 6, no. 2, pp. 81+, 1997. [Online]. Available: http://dx.doi.org/10.1049/esej:19970210

[31] Bratthall, Lars, Johansson, Enrico, and B. Regnell, "Is a Design Rationale Vital when Predicting Change Impact? A Controlled Experiment on Software Architecture Evolution," in *PROFES '00: Proceedings of the Second International Conference on Product Focused Software Process Improvement*. London, UK: Springer-Verlag, 2000, pp. 126–139. [Online]. Available: http://portal.acm.org/citation.cfm?id=713240

[32] Bray, Tim, Paoli, Jean, Sperberg-McQueen, C. M., Maler, Eve, and F. Yergeau, "Extensible markup language (XML) 1.0 (fifth edition)," W3C, Tech. Rep., November 2008. [Online]. Available: http://www.w3.org/TR/REC-xml

[33] Brooks, "No Silver Bullet Essence and Accidents of Software Engineering," *Computer*, vol. 20, no. 4, pp. 10–19, April 1987. [Online]. Available: http://dx.doi.org/10.1109/MC.1987.1663532

[34] Brown, Alan W. and Penedo, Maria H., "An annotated bibliography on integration in software engineering environments," *SIGSOFT Softw. Eng. Notes*, vol. 17, no. 3, pp. 47–55, 1992. [Online]. Available: http://dx.doi.org/10.1145/140938.140944

[35] Broy, Manfred, "Challenges in automotive software engineering," in *ICSE '06: Proceedings of the 28th international conference on Software engineering*. New York, NY, USA: ACM, 2006, pp. 33–42. [Online]. Available: http://dx.doi.org/10.1145/1134285.1134292

[36] H. Brunelière, Cabot, Jordi, C. Clasen, F. Jouault, and J. Bézivin, "Towards Model Driven Tool Interoperability: Bridging Eclipse and Microsoft Modeling Tools," in *Modelling Foundations and Applications*, ser. Lecture Notes in Computer Science, T. Kühne, Selic,, Bran, Gervais,, Marie-Pierre, and F. Terrier, Eds. Berlin, Heidelberg: Springer Berlin / Heidelberg, 2010, vol. 6138, ch. 5, pp. 32–47. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-13595-8_5

[37] Burge, J. E. and Brown, D. C., "An Integrated Approach for Software Design Checking Using Rationale," in *Proc. Design Computing and Cognition Conference*, Cambridge, MA, 2004.

[38] Burge, Janet E., Carroll, John M., McCall, Raymond, and I. Mistrík, *Rationale-Based Software Engineering*, 1st ed. Springer, May 2008. [Online]. Available: http://www.worldcat.org/isbn/354077582X

[39] Buur, J., "A theoretical approach to mechatronics design," Ph.D. dissertation, Technical University of Denmark, 1990.

[40] Capilla, Rafael, Nava, Francisco, and J. C. Dueñas, "Modeling and Documenting the Evolution of Architectural Design Decisions," in *SHARK-ADI '07: Proceedings of the Second Workshop on SHAring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent*. Washington, DC, USA: IEEE Computer Society, 2007, p. 9. [Online]. Available: http://dx.doi.org/http://dx.doi.org/10.1109/SHARK-ADI.2007.9

[41] Capilla, Rafael, Nava, Francisco, S. Pérez, and J. C. Dueñas, "A web-based tool for managing architectural design decisions," *Sw. Eng. Notes*, vol. 31, no. 5, pp. 4+, 2006. [Online]. Available: http://dx.doi.org/10.1145/1163514.1178644

[42] Card, D. N., McGarry, F. E., and Page, G. T., "Evaluating Software Engineering Technologies," *IEEE Trans. Softw. Eng.*, vol. 13, no. 7, pp. 845–851, 1987. [Online]. Available: http://dx.doi.org/10.1109/TSE.1987.233495

[43] Carroll, John M., Alpert, Sherman R., Karat, John, Van Deusen, Mary, and Rosson, Mary B., "Raison d'Etre: capturing design history and rationale in multimedia narratives," in *CHI '94*, 1994, pp. 192–197. [Online]. Available: http://dx.doi.org/10.1145/191666.191741

[44] Chalmers, A. F., *What Is This Thing Called Science?*, 3rd ed. Hackett Pub Co, January 1999. [Online]. Available: http://www.worldcat.org/isbn/0702230936

[45] Chapin, Ned, Hale, Joanne E., Khaled Md, Ramil, Juan F., and Tan, Wui G., "Types of software evolution and software maintenance," *Journal of Software Maintenance*, vol. 13, no. 1, pp. 3–30, January 2001. [Online]. Available: http://portal.acm.org/citation.cfm?id=371701

[46] Christl, Andreas, Koschke, Rainer, and Margaret Anne, "Automated clustering to support the reflexion method," *Information & Software Technology*, vol. 49, no. 3, pp. 255–274, 2007. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2006.10.015

[47] Cicchetti, Antonio, Di Ruscio, Davide, and Pierantonio, Alfonso, "A Metamodel Independent Approach to Difference Representation," *JOT: Journal of Object Technology*, October 2007. [Online]. Available: http://www.jot.fm/issues/issue_2007_10/paper9/index.html

[48] S. Clarke, "Extending standard UML with model composition semantics," *Sci. Comput. Program.*, vol. 44, no. 1, pp. 71–100, July 2002. [Online]. Available: http://dx.doi.org/10.1016/S0167-6423(02)00030-8

[49] Clements, Paul, Bachmann, Felix, Bass, Len, Garlan, David, Ivers, James, Little, Reed, Nord, Robert, and Stafford, Judith, *Documenting Software Architectures: Views and Beyond*. Addison-Wesley Professional, September 2002. [Online]. Available: http://www.worldcat.org/isbn/0201703726

[50] Clements, Paul C., "A Survey of Architecture Description Languages," in *IWSSD '96: Proceedings of the 8th International Workshop on Software Specification and Design*. Washington, DC, USA: IEEE Computer Society, 1996. [Online]. Available: http://portal.acm.org/citation.cfm?id=858261

[51] Clerc, Viktor, Lago, Patricia, and van Vliet, Hans, "Global Software Development: Are Architectural Rules the Answer?" *Global Software Engineering, International Conference on*, vol. 0, pp. 225–234, 2007. [Online]. Available: http://dx.doi.org/10.1109/ICGSE.2007.21

[52] Conklin, E. Jeffrey and Yakemovic, K. C. Burgess, "A process-oriented approach to design rationale," *Hum.-Comput. Interact.*, vol. 6, no. 3, pp. 357–391, 1991. [Online]. Available: http://dx.doi.org/10.1207/s15327051hci0603%5C&4_6

[53] Cook, C. and Visconti, M., "Documentation is important," *CrossTalk*, vol. 7, no. 11, pp. 26–30, 1994.

[54] Crnkovic, Ivica, "Component-based Software Engineering - New Challenges in Software Development," in *Proceedings of the 25th International Conference on Information Technology Interfaces 2003 (ITI 2003)*, vol. 2, 2001, pp. 127–133. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.29.2203

[55] Crnkovic, Ivica, Asklund, Ulf, and Dahlqvist, Annita P., *Implementing and integrating product data management and software configuration management*, ser. Artech House computing library. Artech House, 2003. [Online]. Available: http://www.worldcat.org/isbn/1580534988

[56] J. Cuadrado and J. Molina, "Modularization of model transformations through a phasing mechanism," *Software and Systems Modeling*, 2009. [Online]. Available: http://dx.doi.org/10.1007/s10270-008-0093-0

[57] Cuenot, Philippe, Frey, Patrik, Johansson, Rolf, H. Lönn, Papadopoulos, Yiannis, Reiser, Mark-Oliver, Sandberg, Anders, Servat, David, Kolagari, Ramin T., M. Törngren, and Weber, Matthias, "The EAST-ADL Architecture Description Language for Automotive Embedded Software," *Model-Based Engineering of Embedded Real-Time Systems*, 2010.

[58] Dantas, Cristine R., Murta, Leonardo G. P., and Werner, Claudia M. L., "Consistent Evolution of UML Models by Automatic Detection of Change Traces," in *IWPSE '05: Proceedings of the Eighth International Workshop on Principles of Software Evolution*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 14–147. [Online]. Available: http://dx.doi.org/10.1109/IWPSE.2005.10

[59] Debruyne, Vincent, F. Simonot-Lion, and Trinquet, Yvon, "EAST-ADL: An architecture description language," in *Architecture Description Languages*, ser. IFIP The International Federation for Information Processing. Springer Boston, 2005, ch. 12, pp. 181–195. [Online]. Available: http://dx.doi.org/10.1007/0-387-24590-1_12

[60] Del Fabro, Marcos, J. Bézivin, and Valduriez, Patrick, "Model-Driven Tool Interoperability: An Application in Bug Tracking," in *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE* , ser. Lecture Notes in Computer Science, Meersman, Robert and Tari, Zahir, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, vol. 4275, ch. 53, pp. 863–881. [Online]. Available: http://dx.doi.org/10.1007/11914853_53

[61] Del Fabro, Marcos D., J. Bézivin, F. Jouault, and Breton, Erwan, "AMW: A generic model weaver," in *Premiered Journees sur l'Ingenierie Dirigee par les Modeles*, July 2005. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.108.1294

[62] Del Fabro, Marcos D., J. Bézivin, and Valduriez, Patrick, "Weaving models with the Eclipse AMW plugin," in *In Eclipse Modeling Symposium, Eclipse Summit Europe*, 2006. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.100.7255

[63] Del Fabro, Marcos D. and Valduriez, Patrick, "Towards the efficient development of model transformations using model weaving and matching transformations," *Software and Systems Modeling*, vol. 8, no. 3, pp. 305–324, July 2009. [Online]. Available: http://dx.doi.org/10.1007/s10270-008-0094-z

[64] T. Dingsøyr and van Vliet, Hans, "Introduction to Software Architecture and Knowledge Management," in *Software Architecture Knowledge Management*, Ali Babar, Muhammad, T. Dingsøyr, Lago, Patricia, and Vliet, Hans, Eds. Berlin, Heidelberg: Springer-Verlag, 2009, ch. 1, pp. 1–17. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-02374-3_1

[65] Dodig-Crnkovic, Gordana, "Scientific methods in computer science," in *Conference for the promotion of research in IT*. Department of Computer Science, 2002. [Online]. Available: http://gordana.se/work/cs_method.pdf

[66] Easterbrook, Steve, Singer, Janice, Storey, Margaret-Anne, and Damian, Daniela, "Selecting Empirical Methods for Software Engineering Research," in *Guide to Advanced Empirical Software Engineering*, Shull, Forrest and Singer, Janice, Eds. London: Springer London, 2008, ch. 11, pp. 285–311–311. [Online]. Available: http://dx.doi.org/10.1007/978-1-84800-044-5_11

[67] Eastwood, A., "Firm fires shots at legacy systems," *Computing Canada*, vol. 19, no. 2, p. 17, 1993.

[68] Ebert, Christof and Jones, Capers, "Embedded Software: Facts, Figures, and Future," *Computer*, vol. 42, no. 4, pp. 42–52, April 2009. [Online]. Available: http://dx.doi.org/10.1109/MC.2009.118

[69] Ebert, Christof and Salecker, Juergen, "Embedded Software Technologies and Trends," *IEEE Software*, vol. 26, no. 3, pp. 14–18, 2009. [Online]. Available: http://dx.doi.org/10.1109/MS.2009.70

[70] Egyed, Alexander, "Validating Consistency between Architecture and Design Descriptions," March 2002.

[71] Egyed, Alexander and Medvidovic, Nenad, "Consistent Architectural Refinement and Evolution using the Unified Modeling Language," March 2001.

[72] Ehrig, Karsten, Taentzer, Gabriele, and Varro, Daniel, "Tool integration by model transformations based on the Eclipse Modeling Framework," in *EASST Newsletter*, vol. 12, 2006. [Online]. Available: http://mycite.omikk.bme.hu/doc/16597.pdf

[73] El-khoury, J., Redell, O., and M. Törngren, "A Tool Integration Platform for Multi-Disciplinary Development," in *31st EUROMICRO Conference on Software Engineering and Advanced Applications*. IEEE, 2005, pp. 442–450. [Online]. Available: http://dx.doi.org/10.1109/EUROMICRO.2005.10

[74] El-khoury, Jad, Chen, DeJiu, and M. Törngren, "A Survey of Modelling Approaches for Embedded Computer Control Systems," Royal Institute of Technology, KTH, Stockholm, Sweden, Tech. Rep., 2003.

[75] Eppinger, Steven and Salminen, Vesa, "Patterns of Product Development Interactions," in *INTERNATIONAL CONFERENCE ON ENGINEERING DESIGN ICED 01*, 2001. [Online]. Available: http://dspace.mit.edu/bitstream/handle/1721.1/3808/Eppinger-Salminen_Patterns_WkgPpr_2001.pdf?sequence=2

[76] Erlikh, Len, "Leveraging Legacy System Dollars for E-Business," *IT Professional*, vol. 2, no. 3, pp. 17–23, 2000. [Online]. Available: http://dx.doi.org/http://dx.doi.org/10.1109/6294.846201

[77] Falessi, Davide, Cantone, Giovanni, and Kruchten, Philippe, "Value-Based Design Decision Rationale Documentation: Principles and Empirical Feasibility Study," in *WICSA '08: Proceedings of the Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008)*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 189–198. [Online]. Available: http://dx.doi.org/10.1109/WICSA.2008.8

[78] Falessi, Davide, Capilla, Rafael, and Cantone, Giovanni, "A value-based approach for documenting design decisions rationale: a replicated experiment," in *SHARK*

'08: Proceedings of the 3rd international workshop on Sharing and reusing architectural knowledge. New York, NY, USA: ACM, 2008, pp. 63–70. [Online]. Available: http://doi.acm.org/10.1145/1370062.1370079

[79] J.-R. Falleri, Huchard, Marianne, Lafourcade, Mathieu, and C. Nebut, "Metamodel Matching for Automatic Model Transformation Generation," in *Model Driven Engineering Languages and Systems*, ser. Lecture Notes in Computer Science, Czarnecki, Krzysztof, Ober, Ileana, Bruel, Jean-Michel, Uhl, Axel, and Voelter, Markus, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, vol. 5301, ch. 24, pp. 326–340. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-87875-9_24

[80] Favre, Jean M., "Towards a basic theory to model model driven engineering," in *In Workshop on Software Model Engineering, WISME 2004, joint event with UML2004*, 2004.

[81] Fielding, Roy T., "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[82] Flyvbjerg, Bent, "Five Misunderstandings About Case-Study Research," *Qualitative Inquiry*, vol. 12, no. 2, pp. 219–245, April 2006. [Online]. Available: http://dx.doi.org/10.1177/1077800405284363

[83] France, R., Ray, I., Georg, G., and Ghosh, S., "An aspect-oriented approach to early design modeling," in *IEE Proceedings Software*, vol. 151, no. 4, 2004, pp. 173–185.

[84] France, Robert and Rumpe, Bernhard, "Model-driven Development of Complex Software: A Research Roadmap," in *FOSE '07: 2007 Future of Software Engineering*. Washington, DC, USA: IEEE Computer Society, May 2007, pp. 37–54. [Online]. Available: http://dx.doi.org/10.1109/FOSE.2007.14

[85] Frost, Randall, "Jazz and the Eclipse way of collaboration," *IEEE Software*, vol. 24, no. 6, pp. 114–117, November 2007. [Online]. Available: http://dx.doi.org/10.1109/MS.2007.170

[86] Gajski, Daniel D., Vahid, Frank, Narayan, Sanjiv, and Gong, Jie, *Specification and Design of Embedded Systems*. Prentice Hall PTR, July 1994. [Online]. Available: http://www.worldcat.org/isbn/0131507311

[87] Gamma, E., Helm, R., Johnson, R., and Vlissides, J., *Design Patterns*. Addison-Wesley, 1996.

[88] K. Garcés, F. Jouault, Cointe, Pierre, and J. Bézivin, "A Domain Specific Language for Expressing Model Matching," in *Proceedings of the 5ère Journée sur l'Ingénierie Dirigée par les Modèles (IDM09)*, 2009.

[89] Gero, John S., "Design prototypes: a knowledge representation schema for design," *AI Mag.*, vol. 11, no. 4, pp. 26–36, 1990. [Online]. Available: http://portal.acm.org/citation.cfm?id=95793

[90] Gerstlauer, Andreas and Gajski, Daniel D., "System-Level Abstraction Semantics," in *Proceedings of the International Symposium on System Synthesis*, vol. 15, 2002, pp. 231–236. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.12.9504

[91] Giese, Holger, Hildebrandt, Stephan, and Neumann, Stefan, "Towards Integrating SysML and AUTOSAR Modeling via Bidirectional Model Synchronization," in *MBEES*, 2009, pp. 155–164.

[92] Giese, Holger, Levendovszky, Tihamer, and Vangheluwe, Hans, "Summary of the Workshop on Multi-Paradigm Modeling: Concepts and Tools," in *MoDELS Workshops*, ser. Lecture Notes in Computer Science, T. Kühne, Ed., vol. 4364. Springer, 2006, pp. 252–262. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-69489-2_31

[93] Giese, Holger and Wagner, Robert, "From model transformation to incremental bidirectional model synchronization," *Software and Systems Modeling*, vol. 8, no. 1, pp. 21–43–43, February 2009. [Online]. Available: http://dx.doi.org/10.1007/s10270-008-0089-9

[94] Groher, Iris and Voelter, Markus, "XWeave: models and aspects in concert," in *AOM '07: Proceedings of the 10th international workshop on Aspect-oriented modeling*. New York, NY, USA: ACM, 2007, pp. 35–40. [Online]. Available: http://dx.doi.org/10.1145/1229375.1229381

[95] Gruber, T., "A translation approach to portable ontology specifications," *Knowledge Acquisition*, vol. 5, no. 2, pp. 199–220, June 1993. [Online]. Available: http://dx.doi.org/10.1006/knac.1993.1008

[96] Habli, Ibrahim and Kelly, Tim, "Capturing and Replaying Architectural Knowledge through Derivational Analogy," in *SHARK-ADI '07: Proceedings of the Second Workshop on SHAring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 4+. [Online]. Available: http://dx.doi.org/10.1109/SHARK-ADI.2007.6

[97] Hatchuel, Armand and B. Weil, "A New Approach of Innovative Design: An Introduction to C-K Theory," in *International Conference on Engineering Design (ICED)*, Stockholm, August 2003.

[98] Hein, C., Ritter, T., and Wagner, M., "Model-Driven Tool Integration with ModelBus," in *Workshop Future Trends of Model-Driven Development*, 2009.

[99] Herbsleb, J. D., Paulish, D. J., and Bass, M., "Global software development at Siemens: experience from nine projects," in *Proceedings of the 27th International Conference on Software Engineering*, 2005, pp. 524–533. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1553598

[100] Herrmannsdoerfer, M., "Operation-based versioning of metamodels with COPE," in *CVSM '09: Proceedings of the 2009 ICSE Workshop on Comparison and Versioning of Software Models*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 49–54. [Online]. Available: http://dx.doi.org/10.1109/CVSM.2009.5071722

[101] Herrmannsdoerfer, Markus, Benz, Sebastian, and Juergens, Elmar, "Automatability of Coupled Evolution of Metamodels and Models in Practice," in *MoDELS '08: Proceedings of the 11th international conference on Model Driven Engineering Languages and Systems*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 645–659. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-87875-9_45

[102] ——, "COPE - automating coupled evolution of metamodels and models," in *ECOOP 2009*, ser. Lecture Notes in Computer Science, Drossopoulou, Sophia, Ed., vol. 5653. Berlin, Heidelberg: Springer Berlin / Heidelberg, 2009, pp. 52–76. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-03013-0_4

[103] Herrmannsdoerfer, Markus and Koegel, Maximilian, "Towards a generic operation recorder for model evolution," in *IWMCP '10: Proceedings of the 1st International Workshop on Model Comparison in Practice*. New York, NY, USA: ACM, 2010, pp. 76–81. [Online]. Available: http://dx.doi.org/10.1145/1826147.1826161

[104] Herzog, Erik, "An Approach to Systems Engineering Tool Data Representaton and Exchange," Ph.D. dissertation, Linköping University, 2004.

[105] Hilliard, Rich and Rice, Tim, "Expressiveness in architecture description languages," in *ISAW '98: Proceedings of the third international workshop on Software architecture*. New York, NY, USA: ACM Press, 1998, pp. 65–68. [Online]. Available: http://dx.doi.org/10.1145/288408.288425

[106] Hubka, Vladimir and Eder, W. Ernst, *Theory of Technical Systems: A Total Concept Theory for Engineering Design*. Springer, 1988. [Online]. Available: http://www.worldcat.org/isbn/0387174516

[107] Hubka, Vladimir and Ernst, *Design Science: Introduction to the Needs, Scope and Organization of Organization of Engineering Design Knowledge*, 1st ed. Springer, December 1995. [Online]. Available: http://www.worldcat.org/isbn/3540199977

[108] Huff, S., "Information systems maintenance," *The Business Quarterly*, vol. 50, 1990.

[109] IEEE, "IEEE Standard Glossary of Software Engineering Terminology," IEEE, Tech. Rep., 1990. [Online]. Available: http://dx.doi.org/10.1109/IEEESTD.1990.101064

[110] ——, "Recommended practice for architectural description of software-intensive systems (IEEE Std 1471-2000)," IEEE, Tech. Rep., 2000.

[111] IEEE and ISO/IEC, "Systems and software engineering - Recommended practice for architectural description of software-intensive systems (ISO/IEC 42010 IEEE Std 1471-2000)," ISO, Tech. Rep., July 2007. [Online]. Available: http://dx.doi.org/10.1109/IEEESTD.2007.386501

[112] iFEST Project Partners, "iFEST Project Deliverable D2.2: Requirements for an Integration Platform," ARTEMIS, Tech. Rep., 2010.

[113] ISO, "Industrial automation systems and integration – product data representation and exchange (ISO 10303)," ISO, Tech. Rep., 1994. [Online]. Available: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=20579

[114] Jansen, A. and Bosch, J., "Software Architecture as a Set of Architectural Design Decisions," in *Software Architecture, 2005. WICSA 2005. 5th Working IEEE/IFIP Conference on*. Washington, DC, USA: IEEE, 2005, pp. 109–120. [Online]. Available: http://dx.doi.org/10.1109/WICSA.2005.61

[115] Jansen, Anton, van der Ven, Jan, Avgeriou, Paris, and Hammer, Dieter K., "Tool Support for Architectural Decisions," *Software Architecture, Working IEEE/IFIP Conference on*, vol. 0, pp. 4+, 2007. [Online]. Available: http://dx.doi.org/10.1109/WICSA.2007.47

[116] Jouault, F., Allilaire, F., J. Bézivin, and Kurtev, I., "ATL: a model transformation tool," *Science of Computer Programming*, vol. 72, pp. 31–39, June 2008.

[117] Kappel, Gerti, Kapsammer, Elisabeth, Kargl, Horst, Kramler, Gerhard, Reiter, Thomas, Retschitzegger, Werner, Schwinger, Wieland, and Wimmer, Manuel, "Lifting Metamodels to Ontologies: A Step to the Semantic Integration of Modeling Languages," *Model Driven Engineering Languages and Systems*, vol. 4199, pp. 528–542, 2006. [Online]. Available: http://dx.doi.org/10.1007/11880240_37

[118] Kapsammer, Elisabeth, Kargl, Horst, Kramler, Gerhard, Reiter, Thomas, Retschitzegger, Werner, and Wimmer, Manuel, "On Models and Ontologies - A Layered Approach for Model-based Tool Integration," in *in Proceedings of the Modellierung 2006 (MOD2006*, 2006, pp. 11–27. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.86.3053

[119] Kapsammer, Elisabeth and Reiter, Thomas. Model-Based Tool Integration- State of the Art and Future Perspectives. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.90.1751

[120] Karsai, Gabor, Lang, Andras, and Neema, Sandeep, "Design patterns for open tool integration," *Software and Systems Modeling*, vol. 4, no. 2, pp. 157–170, May 2005. [Online]. Available: http://dx.doi.org/10.1007/s10270-004-0073-y

[121] Karsenty, Laurent, "An empirical evaluation of design rationale documents," in *CHI '96: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA: ACM, 1996, pp. 150–156. [Online]. Available: http://doi.acm.org/10.1145/238386.238462

[122] Kelly, Steven and Tolvanen, Juha-Pekka, *Domain-Specific Modeling: Enabling Full Code Generation*. Wiley-IEEE Computer Society Pr, March 2008. [Online]. Available: http://www.worldcat.org/isbn/0470036664

[123] Kelly, T. P., "Arguing safety – a systematic approach to managing safety cases," Ph.D. dissertation, Department of Computer Science, University of York, 1999.

[124] Kern, Heiko, "The Interchange of (Meta)Models between MetaEdit+ and Eclipse EMF Using M3-Level-Based Bridges," in *8th OOPSLA Workshop on Domain-Specific Modeling at OOPSLA 2008*, Gray, Jeff, Sprinkle, Jonathan, Tolvanen, Juha-Pekka, and Rossi, Matti, Eds. University of Alabama at Birmingham, 2008, pp. 14–19.

[125] Kern, Heiko and S. Kühne, "Model Interchange between ARIS and Eclipse EMF," in *Proceedings of the 7th OOPSLA Workshop on Domain-Specific Modeling (DSM'07)*, ser. Computer Science and Information System Reports, Technical Reports, Tolvanen, Juha-Pekka, Gray, Jeff, Rossi, Matti, and Sprinkle, Jonathan, Eds., no. TR-38. Finland: University of Jyväskylä, 2007, pp. 105–114.

[126] Kern, Heiko and Kuhne, Stefan, "Integration of Microsoft Visio and EclipseModeling Framework Using M3-Level-Based Bridges," in *2nd Workshop on Model-Driven Tool & Process Integration (MDTPI 2009) at the European Conference on Modeling Foundations and Applications (ECMFA 2009)*, June 2009.

[127] Khoury, Jad E., "A Model Management and Integration Platform for Mechatronics Product Development," Ph.D. dissertation, KTH, 2006.

[128] Kiczales, Gregor, Lamping, John, Menhdhekar, Anurag, Maeda, Chris, Lopes, Cristina, Loingtier, Jean M., and Irwin, John, "Aspect-Oriented Programming," in *Proceedings European Conference on Object-Oriented Programming*, M. Akşit and Matsuoka, Satoshi, Eds. Berlin, Heidelberg, and New York: Springer-Verlag, 1997, vol. 1241, pp. 220–242. [Online]. Available: citeseer.ist.psu.edu/kiczales97aspectoriented.html

[129] Knuth, Donald E., "backus normal form vs. Backus Naur form," *Commun. ACM*, vol. 7, no. 12, pp. 735–736, 1964. [Online]. Available: http://dx.doi.org/10.1145/355588.365140

[130] A. Königs and A. Schürr, "Tool Integration with Triple Graph Grammars - A Survey," *Electron. Notes Theor. Comput. Sci.*, vol. 148, no. 1, pp. 113–150, February 2006. [Online]. Available: http://dx.doi.org/10.1016/j.entcs.2005.12.015

[131] Koschke, Rainer and Simon, Daniel, "Hierarchical Reflexion Models," in *Working Conference on Reverse Engineering*. IEEE Computer Society Press, November 2003, pp. 36–45.

[132] Kramler, G., Kappel, G., Reiter, T., Kapsammer, E., Retschitzegger, W., and Schwinger, W., "Towards a semantic infrastructure supporting model-based tool integration," in *GaMMa '06: Proceedings of the 2006 international workshop on Global integrated model management*. New York, NY, USA: ACM, 2006, pp. 43–46. [Online]. Available: http://dx.doi.org/10.1145/1138304.1138314

[133] Kruchten, Philippe, "An Ontology of Architectural Design Decisions in Software Intensive Systems," in *2nd Groningen Workshop Software Variability*, October 2004, pp. 54–61.

[134] Kruchten, Philippe, Capilla, Rafael, and J. C. Dueñas, "The Decision View's Role in Software Architecture Practice," *IEEE Softw.*, vol. 26, no. 2, pp. 36–42, 2009. [Online]. Available: http://dx.doi.org/10.1109/MS.2009.52

[135] Kum, Daehyun, Park, Gwang-Min, Lee, Seonghun, and Jung, Wooyoung, "AUTOSAR migration from existing automotive software," in *Control, Automation and Systems, 2008. ICCAS 2008. International Conference on*, October 2008, pp. 558–562. [Online]. Available: http://dx.doi.org/10.1109/ICCAS.2008.4694565

[136] Kunz, W. and Rittel, H. W. J., "Issues as Elements of Information Systems," Univ. Calif. at Berkeley, Tech. Rep., 1970.

[137] Kurtev, Ivan, J. Bézivin, and Aksit, Mehmet, "Technological spaces: An initial appraisal," in *CoopIS, DOA 2002 Federated Conferences, Industrial track*, 2002. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.109.332

[138] Lee, Jintae, "Decision representation language (DRL) and its support environment," MIT Artificial Intelligence Laboratory, Tech. Rep., August 1989. [Online]. Available: http://hdl.handle.net/1721.1/41499

[139] Lehman, M. M. and Belady, L. A., Eds., *Program evolution: processes of software change*. San Diego, CA, USA: Academic Press Professional, Inc., 1985.

[140] Leveson, Nancy, *Safeware : System Safety and Computers*. Addison-Wesley, April 1995. [Online]. Available: http://www.worldcat.org/isbn/0201119722

[141] Leveson, Nancy G., "Intent Specifications: An Approach to Building Human-Centered Specifications," *IEEE Transactions on Sw. Eng.*, vol. 26, no. 1, pp. 15–35, January 2000.

[142] Leveson, Nancy G. and Weiss, Kathryn A., "Making embedded software reuse practical and safe," in *SIGSOFT '04/FSE-12: Proceedings of the 12th ACM SIGSOFT twelfth international symposium on Foundations of software engineering*. New York, NY, USA: ACM Press, 2004, pp. 171–178. [Online]. Available: http://dx.doi.org/10.1145/1029894.1029897

[143] Lientz, Bennet P., *Software Maintenance Management: A Study of the Maintenance of Computer Application Software in 487 Data Processing Organizations*. Addison-Wesley Pub (Sd), 1980. [Online]. Available: http://www.worldcat.org/isbn/0201042053

[144] Liggesmeyer, Peter and Trapp, Mario, "Trends in Embedded Software Engineering," *Software, IEEE*, vol. 26, no. 3, pp. 19–25, April 2009. [Online]. Available: http://dx.doi.org/10.1109/MS.2009.80

[145] MacLean, Allan, Young, Richard M., Bellotti, Victoria M. E., and Moran, Thomas P., "Questions, options, and criteria: elements of design space analysis," *Human-Computer Interaction*, vol. 6, no. 3, pp. 53–105, 1996. [Online]. Available: http://portal.acm.org/citation.cfm?id=261707

[146] McCall, R., "PHI: a conceptual foundation for design hypermedia," *Design Studies*, vol. 12, no. 1, pp. 30–41, January 1991. [Online]. Available: http://dx.doi.org/10.1016/0142-694X(91)90006-I

[147] Medvidovic, Nenad, Dashofy, Eric M., and Taylor, Richard N., "Moving architectural description from under the technology lamppost," *Information and Software Technology*, vol. 49, no. 1, pp. 12–31, January 2007. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2006.08.006

[148] Medvidovic, Nenad and Taylor, Richard N., "A Classification and Comparison Framework for Software Architecture Description Languages," *IEEE Transactions on Software Engineering*, vol. 26, no. 1, pp. 70–93, 2000. [Online]. Available: http://dx.doi.org/10.1109/32.825767

[149] Melnik, Sergey, Rahm, Erhard, and Bernstein, Philip A., "Rondo: a programming platform for generic model management," in *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM, 2003, pp. 193–204. [Online]. Available: http://dx.doi.org/10.1145/872757.872782

[150] Mens, Tom, R. Van Der Straeten, and D'Hondt, Maja, "Detecting and Resolving Model Inconsistencies Using Transformation Dependency Analysis," in *Proc. Int'l Conf. MoDELS 2006*, ser. LNCS, vol. 4199. Springer-Verlag, October 2006, pp. 200–214.

[151] Mens, Tom and Van Gorp, Pieter, "A Taxonomy of Model Transformation," *Electr. Notes Theor. Comput. Sci*, vol. 152, pp. 125–142, 2006.

[152] ——, "A Taxonomy of Model Transformation," *Electr. Notes Theor. Comput. Sci*, vol. 152, pp. 125–142, 2006.

[153] Mernik, Marjan, Heering, Jan, and Sloane, Anthony M., "When and how to develop domain-specific languages," *ACM Comput. Surv.*, vol. 37, no. 4, pp. 316–344, December 2005. [Online]. Available: http://dx.doi.org/10.1145/1118890.1118892

[154] MODELWARE Project, "MODELWARE D5.3-1 Industrial ROI, Assessment, and Feedback- Master Document." MODELWARE, http://www.modelware-ist.org, Tech. Rep., 2006.

[155] Mohagheghi, Parastoo and Dehlen, Vegard, "Where Is the Proof? - A Review of Experiences from Applying MDE in Industry," in *Model Driven Architecture - Foundations and Applications*, ser. Lecture Notes in Computer Science, Schieferdecker, Ina and Hartman, Alan, Eds. Berlin, Heidelberg: Springer Berlin / Heidelberg, 2010, vol. 5095, ch. 31, pp. 432–443. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-69100-6_31

[156] Murphy, G. C., Notkin, D., and Sullivan, K. J., "Software reflexion models: bridging the gap between design and implementation," *IEEE Transactions on Software Engineering*, vol. 27, no. 4, 2001.

[157] Muskens, J., Bril, R. J., and Chaudron, M. R. V., "Generalizing Consistency Checking between Software Views," *Software Architecture, 2005. WICSA 2005. 5th Working IEEE/IFIP Conference on*, pp. 169–180, 2005. [Online]. Available: http://dx.doi.org/10.1109/WICSA.2005.37

[158] Narayanan, Anantha, Levendovszky, Tihamer, Balasubramanian, Daniel, and Karsai, Gabor, "Automatic Domain Model Migration to Manage Metamodel Evolution," in *Model Driven Engineering Languages and Systems*, 2009, pp. 706–711. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-04425-0_57

[159] Navarro, Elena and Cuesta, Carlos, "Automating the Trace of Architectural Design Decisions and Rationales Using a MDD Approach," in *Software Architecture*, ser. Lecture Notes in Computer Science, Morrison, Ron, Balasubramaniam, Dharini, and Falkner, Katrina, Eds. Berlin, Heidelberg: Springer Berlin / Heidelberg, 2008, vol. 5292, ch. 10, pp. 114–130–130. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-88030-1_10

[160] Navarro, Israel, Lundqvist, Kristina, and Leveson, Nancy. An Intent Specification Model for a Robotic Software Control System. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.70.7351

[161] Navet, Nicolas and Simonot-Lion, Francoise, Eds., *Automotive Embedded Systems Handbook*. CRC Press, 2008.

[162] Nentwich, Christian, Emmerich, Wolfgang, and Finkelstein, Anthony, "Flexible Consistency Checking," *ACM Transactions on Software Engineering and Methodology*, vol. 12, no. 1, pp. 28–63, January 2003. [Online]. Available: citeseer.ist.psu.edu/nentwich01flexible.html

[163] Noy, Natalya F., "Semantic integration: a survey of ontology-based approaches," *SIGMOD Rec.*, vol. 33, no. 4, pp. 65–70, December 2004. [Online]. Available: http://dx.doi.org/10.1145/1041410.1041421

[164] Oliver, David W., Kelliher, Timothy P., and James, *Engineering Complex Systems With Models and Objects*. McGraw-Hill Companies, 1997. [Online]. Available: http://www.worldcat.org/isbn/0070481881

[165] OMG, "Model Driven Architecture (MDA) Guide," OMG, Tech. Rep., 2003. [Online]. Available: http://www.omg.org/mda/

[166] ——, "Meta Object Facility (MOF), v2.0," OMG, Tech. Rep., January 2006. [Online]. Available: http://www.omg.org/spec/MOF/2.0/

[167] ——, "MOF 2.0 / XMI Mapping Specification, v2.1.1," OMG, Tech. Rep., December 2007. [Online]. Available: http://www.omg.org/technology/documents/formal/xmi.htm

[168] ——, "MOF 2.0 Query / View / Transformation," OMG, Tech. Rep., December 2009. [Online]. Available: http://www.omg.org/spec/QVT

[169] ——, "Systems Modeling Language (SysML)," OMG, Tech. Rep., 2010. [Online]. Available: http://www.omgsysml.org/

[170] ——. (2010) Unified Modeling Language (UML). [Online]. Available: http://www.omg.com/uml/

[171] OSLC Core Specification Workgroup, "OSLC core specification version 2.0," Open Services for Lifecycle Collaboration, Tech. Rep., August 2010.

[172] Pahl, G., Beitz, W., Feldhusen, J., and Grote, K. H., *Engineering Design: A Systematic Approach*. Springer, January 2007. [Online]. Available: http://www.worldcat.org/isbn/1846283183

[173] Parnas, David L., "Software Aging," in *ICSE*, 1994, pp. 279–287. [Online]. Available: http://portal.acm.org/citation.cfm?id=257734.257788

[174] Pence, J. and Hon III, "Building software quality into telecommunications network systems," *Quality Progress*, pp. 95–97, October 1993.

[175] Perry, D. and Wolf, A., "Foundations for the Study of Software Architecture," *ACM SIGSOFT Software Engineering Notes*, vol. 17, no. 4, pp. 40–52, October 1992.

[176] Pilato, C. Michael, Collins-Sussman, Ben, and Fitzpatrick, Brian W., *Version Control with Subversion*, 1st ed. O'Reilly Media, June 2004. [Online]. Available: http://www.worldcat.org/isbn/0596004486

[177] A. Postma, "A method for module architecture verification and its application on a large component-based system," *Information & Software Technology*, vol. 45, no. 4, pp. 171–194, 2003.

[178] Pretschner, Alexander, Broy, Manfred, Kruger, Ingolf H., and Stauner, Thomas, "Software Engineering for Automotive Systems: A Roadmap," in *FOSE '07: 2007 Future of Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 55–71. [Online]. Available: http://dx.doi.org/10.1109/FOSE.2007.22

[179] Reiser, M. O., "Managing Complex Variability in Automotive Software Product Lines with Subscoping and Configuration Links," Ph.D. dissertation, TU Berlin, 2008.

[180] Reiter, Thomas, Altmanninger, Kerstin, and Retschitzegger, Werner, "Think Global, Act Local: Implementing Model Management with Domain-Specific Integration Languages," in *Models in Software Engineering*, ser. Lecture Notes in Computer Science, T. Kühne, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, vol. 4364, ch. 32, pp. 263–276. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-69489-2_32

[181] Rombach, H. and Basili, V., "Quantitative assessment of maintenance: an industrial case study," in *Proceedings of the IEEE Conference on Software Maintenance*, September 1987, pp. 134–144. [Online]. Available: http://www.cs.umd.edu/users/basili/publications/proceedings/P40.pdf

[182] Rose, Louis M., Paige, Richard F., Kolovos, Dimitrios S., and Polack, Fiona A. C., "An Analysis of Approaches to Model Migration," in *Proc. Models and Evolution (MoDSE-MCCM) Workshop, 12th ACM/IEEE International Conference on Model Driven Engineering, Languages and Systems*, October 2009.

[183] Rozanski, Nick and Woods, Eoin, *Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives*. Addison-Wesley Professional, April 2005. [Online]. Available: http://www.worldcat.org/isbn/0321112296

[184] Salzmann, Christian and Stauner, Thomas, "Automotive Software Engineering," in *Languages for System Specification*, Grimm,, Christoph, Ed. Boston: Springer US, 2004, ch. 21, pp. 333–347. [Online]. Available: http://dx.doi.org/10.1007/1-4020-7991-5_21

[185] A. Schürr and H. Dörr, "Introductory paper," *Software and Systems Modeling*, vol. 4, no. 2, pp. 109–111, May 2005. [Online]. Available: http://dx.doi.org/10.1007/s10270-004-0069-7

[186] Schuster, N., Zimmermann, O., and Pautasso, C., "ADkwik: Web 2.0 Collaboration System for Architectural Decision Engineering," in *SEKE*, 2007, pp. 255–260.

[187] Sellgren, Ulf, M. Törngren, Malvius, Diana, and Biehl, Matthias, "PLM for Mechatronics Integration," in *Proceedings of the 6th International Product Lifecycle Management Conference (PLM 2009)*, July 2009.

[188] Seng, Olaf, Bauer, Markus, Biehl, Matthias, and Pache, Gert, "Search-based Improvement of Subsystem Decompositions," in *Proceedings of the 2005 ACM Conference on Genetic and Evolutionary Computation (GECCO 2005)*, 2005, pp. 1045–1051. [Online]. Available: http://dx.doi.org/10.1145/1068009.1068186

[189] Shaw, Mary, "What makes good research in software engineering?" *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 4, no. 1, pp. 1–7–7, October 2002. [Online]. Available: http://dx.doi.org/10.1007/s10009-002-0083-4

[190] Shum, S. J., "A Cognitive Analysis of Design Rationale Representation," Ph.D. dissertation, University of York, 1991. [Online]. Available: http://people.kmi.open.ac.uk/sbs/research/phd/phd.html

[191] Shum, Simon B., "Analyzing the Usability of a Design Rationale Notation," in *in Design Rationale: Concepts, Techniques, and*, 1996, pp. 185–215. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.52.5956

[192] Sprinkle, Jonathan M., "Metamodel driven model migration," Ph.D. dissertation, Vanderbilt University, 2003. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.102.2044&rep=rep1&type=pdf

[193] Star, Susan L. and Griesemer, James R., "Institutional Ecology, 'Translations' and Boundary Objects: Amateurs and Professionals in Berkeley's Museum of Vertebrate Zoology, 1907-39," *Social Studies of Science*, vol. 19, no. 3, pp. 387–420, 1989. [Online]. Available: http://dx.doi.org/10.2307/285080

[194] Steinberg, David, Budinsky, Frank, Paternostro, Marcelo, and Merks, Ed, *EMF: Eclipse Modeling Framework (2nd Edition)*, 2nd ed. Addison-Wesley Professional, January 2008. [Online]. Available: http://www.worldcat.org/isbn/0321331885

[195] Suh, Nam P., *The Principles of Design (Oxford Series on Advanced Manufacturing)*. Oxford University Press, March 1990. [Online]. Available: http://www.worldcat.org/isbn/0195043456

[196] ——, *Axiomatic Design: Advances and Applications (The Oxford Series on Advanced Manufacturing)*. Oxford University Press, May 2001. [Online]. Available: http://www.worldcat.org/isbn/0195134664

[197] Tang, Antony, Babar, Muhammad A., Gorton, Ian, and Han, Jun, "A Survey of the Use and Documentation of Architecture Design Rationale," in *WICSA '05: Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 89–98. [Online]. Available: http://dx.doi.org/10.1109/WICSA.2005.7

[198] ——, "A survey of architecture design rationale," *J. Syst. Softw.*, vol. 79, no. 12, pp. 1792–1804, December 2006. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2006.04.029

[199] Tang, Antony, Jin, Yan, and Han, Jun, "A rationale-based architecture model for design traceability and reasoning," *Journal of Systems and Software*, vol. 80, no. 6, pp. 918–934, June 2007. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2006.08.040

[200] Tang, Antony, Nicholson, Ann, Jin, Yan, and Han, Jun, "Using Bayesian belief networks for change impact analysis in architecture design," *Journal of Systems and Software*, vol. 80, no. 1, pp. 127–148, January 2007. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2006.04.004

[201] Thomas, Ian and Nejmeh, Brian A., "Definitions of Tool Integration for Environments," *IEEE Softw.*, vol. 9, no. 2, pp. 29–35, 1992. [Online]. Available: http://dx.doi.org/10.1109/52.120599

[202] Tichy, W. F., "Should computer scientists experiment more?" *Computer*, vol. 31, no. 5, pp. 32–40, May 1998. [Online]. Available: http://dx.doi.org/10.1109/2.675631

[203] TIMMO Project. (2009) Timing model (TIMMO). [Online]. Available: http://www.timmo.org

[204] M. Törngren, "Towards an industrial Framework for Embedded systems tools," in *First Workshop on Hands-on Platforms and tools for model-based engineering of Embedded Systems (HoPES'10)*, June 2010.

[205] M. Törngren, Chen, DeJiu, and Crnkovic, Ivica, "Component-based vs. Model-based Development: A Comparison in the Context of Vehicular Embedded Systems," in *EUROMICRO '05: Proceedings of the 31st EUROMICRO Conference on Software Engineering and Advanced Applications*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 432–441. [Online]. Available: http://dx.doi.org/10.1109/EUROMICRO.2005.18

[206] Tran, John B., Godfrey, Michael W., Lee, Eric H. S., and Holt, Richard C., "Architectural Repair of Open Source Software," in *In Proceedings of International Workshop on Program Comprehension*, 2000, pp. 48–59.

[207] ——, "Architectural Repair of Open Source Software," in *In Proceedings of International Workshop on Program Comprehension*, 2000, pp. 48–59. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.35.9245

[208] Tratt, Laurence, "Model transformations and tool integration," *Software and Systems Modeling*, vol. 4, no. 2, pp. 112–122, May 2005. [Online]. Available: http://dx.doi.org/10.1007/s10270-004-0070-1

[209] Tyree, Jeff and Akerman, Art, "Architecture Decisions: Demystifying Architecture," *IEEE Software*, vol. 22, no. 2, pp. 19–27, 2005. [Online]. Available: http://dx.doi.org/10.1109/MS.2005.27

[210] Ueda, K., "Synthesis and emergence - research overview," *Artificial Intelligence in Engineering*, vol. 15, no. 4, pp. 321–327, October 2001. [Online]. Available: http://dx.doi.org/10.1016/S0954-1810(01)00022-X

[211] Ulrich, Karl and Eppinger, Steven, *Product Design and Development*, 3rd ed. McGraw-Hill/Irwin, July 2003. [Online]. Available: http://www.worldcat.org/isbn/0072471468

[212] VDI, "Methodic development of solution principles (VDI 2222)," VDI, Tech. Rep., 1997.

[213] ——, "Design methodology for mechatronic systems (VDI 2206)," VDI, Tech. Rep., 2004.

[214] Vesperman, Jennifer, *Essential CVS*, 1st ed. O'Reilly Media, June 2003. [Online]. Available: http://www.worldcat.org/isbn/0596004591

[215] Visconti, Marcello and Cook, Curtis R., "Assessing the State of Software Documentation Practices," in *Product Focused Software Process Improvement*, 2004, pp. 485–496. [Online]. Available: http://www.springerlink.com/content/6dv8y2xgp5n31x0q

[216] Vliet, Hans, Avgeriou, Paris, Boer, Remco C., Clerc, Viktor, Farenhorst, Rik, Jansen, Anton, and Lago, Patricia, "The GRIFFIN Project: Lessons Learned," in *Software Architecture Knowledge Management*, 2009, ch. 8, pp. 137–154. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-02374-3_8

[217] Wache, H., T. Vögele, Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H., and S. Hübner, "Ontology-Based Information Integration: A Survey." [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.12.7857

[218] Wachsmuth, Guido, "Metamodel Adaptation and Model Co-adaptation," in *ECOOP 2007 Object-Oriented Programming*, 2007, pp. 600–624. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-73589-2_28

[219] Wasserman, Anthony I., "Tool Integration in Software Engineering Environments," in *Software Engineering Environments, International Workshop on Environments Proceedings*, ser. Lecture Notes in Computer Science, Long, Fred, Ed. Springer-Verlag, September 1989, pp. 137–149. [Online]. Available: http://www.springerlink.com/content/p582q2n825k87nl5/

[220] Weiss, Kathryn A., Ong, Elwin C., and Leveson, Nancy G., "Reusable specification components for model-driven development," in *In Proceedings of the International Conference on System Engineering, INCOSE*, 2003. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.5.7488

[221] Wicks, M. and Dewar, R., "A new research agenda for tool integration," *Journal of Systems and Software*, vol. 80, no. 9, pp. 1569–1585, September 2007. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2007.03.089

[222] Wicks, M. N., "Tool Integration within Software Engineering Environments: An Annotated Bibliography," Heriot-Watt University, Tech. Rep., 2006. [Online]. Available: http://www.macs.hw.ac.uk:8080/techreps/docs/files/HW-MACS-TR-0041.pdf

[223] Wiegand, John, "The Case for Open Services," IBM Rational Software, Tech. Rep., May 2009.

[224] Woods, Eoin and Hilliard, Rich, "Architecture Description Languages in Practice Session Report," in *WICSA '05: Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 243–246. [Online]. Available: http://dx.doi.org/10.1109/WICSA.2005.15

[225] Wu, Weihang and Kelly, Tim, "Managing Architectural Design Decisions for Safety-Critical Software Systems," in *Quality of Software Architectures* , ser. Lecture Notes in Computer Science, Hofmeister, Christine, Crnkovic, Ivica, and Reussner, Ralf, Eds. Springer Berlin Heidelberg, 2006, vol. 4214, ch. 9, pp. 59–77. [Online]. Available: http://dx.doi.org/10.1007/11921998_9

[226] Zelkowitz, Marvin, *Principles of Software Engineering and Design (Prentice-Hall software series)*. Prentice Hall, 1979. [Online]. Available: http://www.worldcat.org/isbn/013710202X

[227] ——, "An update to experimental models for validating computer technology," *Journal of Systems and Software*, vol. 82, no. 3, pp. 373–376, March 2009. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2008.06.040