A Modeling Language for the Description and Development of Tool Chains for Embedded Systems

MATTHIAS BIEHL

Doctoral Thesis
Stockholm, Sweden, 2013
Abstract

The development of embedded systems is typically supported by a number of diverse development tools. To achieve seamless tool support throughout the embedded systems development process, tool chains are constructed as software solutions that integrate the development tools. Tool chains have grown from ad-hoc solutions to complex software systems, since they need to support distributed engineering, integration conventions, a specific set of tools and the complete product development process used in a company. In practice, the development of tool chains that fulfill these needs is difficult and time-consuming, since it is a largely unsupported, manual engineering task. In addition, tool chains are typically described using general purpose modeling languages or languages borrowed from other domains, which contributes to the accidental complexity of tool chain development. Due to the increasing sophistication and size of tool chains, there is a need for a systematic, targeted description and development approach for tool chains.

This thesis contributes with a language for the systematic description of tool chains and semi-automated techniques to support their development.

- The Tool Integration Language (TIL) is a domain-specific modeling language (DSML) for tool chains that allows describing tool chains explicitly, systematically and at an appropriate level of abstraction. TIL concepts are from the domain of tool integration and express the essential design decisions of tool chains at an architectural level of abstraction. A TIL model serves as a basis for the development of a tailored tool chain.

- Semi-automated techniques for the specification, analysis and synthesis support the development of tool chains that are described as TIL models. Specification techniques support the creation and refinement of a tool chain model that is aligned to a given development process and set of tools. Domain-specific analysis techniques are used to check the alignment of the tool chain model with the supported process. Synthesis techniques support the efficient realization of the specified tool chain model as a software solution that conforms to integration conventions.

Experiences from case studies are presented which apply TIL to support the creation of tool chains. The approach is evaluated, both qualitatively and quantitatively, by comparing it to traditional development methods for tool chains. The approach enables the efficient development of tailored tool chains, which have the potential to improve the productivity of embedded systems development.

Keywords: Tool Chain, Tool Integration, Domain-Specific Modeling Language, Generative Approach, Lifecycle Support for Embedded System Development
Acknowledgements

This thesis would not have been possible without the support of many people. I would like to thank especially:

- Martin Törngren for the opportunity to work towards the doctorate thesis and for providing advise, feedback, encouragement and enthusiasm throughout the work!

- Jad El-khoury for the advise, detailed feedback, our regular planning meetings at KTH and various coffee places in town and for not forgetting what it is like to be a student!

- Frédéric Loiret, Fredrik Asplund, Carl-Johan Sjöstedt, Jiarui Hong and Weiquing Gu for the fruitful discussion and contributions to the appended co-authored papers.

- De-Jiu Chen, Tahir Naseer Qureshi, Magnus Persson, Lei Feng and Sagar Behere for the great working atmosphere and discussions.

- Vicki Derbyshire for the feedback on the writing style.

- All colleagues in the department of machine design and in the research projects ATESST2, CESAR, iFEST and MAENAD.

- ARTEMIS JU, EU Commission and the Wallenberg Stiftelse for funding parts of this work.

- My friends and family, especially Diana, Gertrud, Günter, Vera and Lena for their moral support during the intense periods of the work.

Matthias Biehl
Stockholm, January 2013
# Contents

Acknowledgements ........................................... v

Contents ....................................................... vii

List of Appended Papers .................................. xi

List of Additional Related Papers by the Author ....... xv

I Getting Started ........................................... 1

1 Introduction ................................................. 3
   1.1 Embedded Systems Development ....................... 3
   1.2 Problem Formulation ..................................... 5
   1.3 High-Level Goals ......................................... 9
   1.4 Delimitations and Alternatives ....................... 9
   1.5 Outline .................................................. 10

2 Research Questions and Methods ....................... 13
   2.1 Methodological Frameworks ............................ 13
   2.2 Research Questions and Methods .................... 14

3 Criteria for the Description and Development of Tool Chains ............................................ 19
   3.1 Criteria for the Description of Tool Chains ........ 20
   3.2 Criteria for the Development of Tool Chains ....... 23
   3.3 Summary ................................................ 26

4 State of the Art ........................................... 27
   4.1 Basic Concepts and Terminology ...................... 27
   4.2 Tool Integration ......................................... 30
   4.3 Characterization of Tool Integration Approaches .... 35
II Overview of Results and Evaluation

5 A Modeling Language for Tool Chains
  5.1 Objectives for the Tool Integration Language
  5.2 Domain Analysis for Tool Chains
  5.3 The Tool Integration Language
  5.4 Proposed Tool Chain Development Process with TIL

6 Support for the Development of Tool Chains
  6.1 A Development Environment for TIL
  6.2 Construction of a Conceptual TIL Model
  6.3 Refinement of ToolAdapters in a TIL Model
  6.4 Refinement of DataChannels in a TIL Model
  6.5 Analysis of TIL Models
  6.6 Code Generation from TIL Models

7 Evaluation of TIL
  7.1 TIL for the Description of Tool Chains
  7.2 TIL for the Development of Tool Chains

III Reflection

8 Discussion
  8.1 Primary and Derivative Integration Challenge
  8.2 From Model-Based to Model-Driven Tool Integration
  8.3 Roles of Tools in the Tool Chain
  8.4 Potential Implications for Industrial Practice

9 Conclusion
  9.1 Review of Research Questions
  9.2 Future Work

Bibliography

IV Appended Papers

A A Modular Tool Integration Approach - Experiences from two Case Studies

B A Domain Specific Language for Generating Tool Integration Solutions

C On the Modeling and Generation of Service-Oriented Tool Chains
CONTENTS

D  High-Level Specification and Code Generation for Service-Oriented Tool Adapters  159
E  Constructing Tool Chains based on SPEM Process Models  169
F  Model-based Service Discovery and Orchestration for OSLC Services in Tool Chains  179
G  Automated Construction of Data Integration Solutions for Tool Chains  191
H  Early Automated Verification of Tool Chain Design  203
I  A Cost-Efficiency Model for Tool Chains  217
List of Appended Papers

This thesis is based on the following 9 papers, which are appended to this thesis.

Paper A


Martin provided feedback; 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.

Paper B


Jad, Frederic and Martin provided feedback; Matthias proposed the ideas, performed the work and wrote the paper.

Paper C


Jad, Frederic and Martin provided feedback; Matthias proposed the ideas, performed the work and wrote the paper.
Paper D

Martin provided feedback; Jad and Matthias developed the initial concepts; Matthias implemented the generator, the case study and wrote the paper.

Paper E

Martin provided feedback; Matthias proposed the ideas, performed the work and wrote the paper.

Paper F

Wenqing implemented the discovery algorithm, performed the case study and wrote section 4, Frederic contributed with knowledge on SCA and contributed to section 6, Matthias proposed the idea of OSLC-based discovery, the conceptual design of orchestration and discovery, initiated and led the work, and wrote the rest of the paper.

Paper G

Jiarui implemented the similarity flooding and matching algorithms. Frederic provided feedback. Matthias proposed the ideas, wrote the paper and implemented
the higher-order model transformation and the TIL code generators.

**Paper H**


Matthias proposed the ideas, performed the work and wrote the paper.

**Paper I**


Martin provided feedback; Matthias proposed the ideas, performed the work and wrote the paper.
List of Additional Related Papers by the Author

The following papers are related to this thesis, but are not included.

LIST OF ADDITIONAL RELATED PAPERS BY THE AUTHOR


• CESAR Project Deliverable D_SP5_R3.2 [Online].

• iFEST Project Deliverables D2.1, D2.2, D2.3, D7.2 [Online].
Part I

Getting Started
Chapter 1

Introduction

This work proposes and evaluates a modeling language for the systematic description and development of tool chains, which are used in the context of embedded systems development. Embedded systems development is typically supported by a large number and wide variety of heterogeneous tools. The tools are isolated and it is thus difficult to create a seamless engineering environment using the tools. A tool chain, is such an engineering environment that is realized as a software system. In current practices, the development of tool chains is ad-hoc, lacking an engineering approach with an appropriate description of the tool chain and support for carrying out a systematic development process.

In this chapter, the development of embedded systems is briefly introduced (Section 1.1), since it provides the context for tool integration. The challenges of tool integration, the challenges of tool chain development and the involved stakeholders are identified (Section 1.2). For addressing these challenges, a number of high-level goals for the development of tool chains are set (Section 1.3). Some delimitations are made (Section 1.4) and the outline of this work is described (Section 1.5).

1.1 Embedded Systems Development

In this section, the development of embedded systems is briefly introduced, since it provides the context for the challenges addressed in this thesis. Of particular interest in this introduction are the tools used for embedded systems development.

Many innovative functions in traditionally mechanical systems, such as advanced driver assistance systems in cars or autopilots in airplanes, are realized as embedded computer 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” [91]. In the following, the shortened term *embedded system* is used to refer to an embedded computer system. An embedded system is characterized by heterogeneity, since it consists of tightly connected hardware and software. In addition, the embedded system is integrated into a larger system, e.g.
a traditionally mechanical system such as a car or airplane. To capture the heterogeneous nature and the various properties of the system, it is typically described using multiple views. A view is a work product that expresses a system from a certain perspective [93]. The views are expressed using various languages and are supported by various tools for design, analysis and simulation.

The development of embedded systems involves engineers from different disciplines, e.g. control engineers, mechanical engineers, electrical engineers and software engineers [48]. Each discipline views the product from a different perspective [56], resulting in a number of discipline-specific views, which are supported by discipline-specific development tools.

Industrial development processes for embedded systems typically follow the V-Model described in VDI 2206 [160]. According to VDI 2206, hardware and software are developed separately with certain synchronization points, after the overall requirements have been initially established. While hardware development typically follows a waterfall approach, software development may apply agile or iterative approaches. In addition, approaches for complexity management [155] are increasingly used, such as model-based/model-driven development [131] and component-based development [62].

1.1.1 Tools for Embedded Systems Development

The activities in embedded systems development are increasingly supported by development tools, and many alternative tools are emerging, leading in turn to a broad and heterogeneous landscape of development tools.

The different phases of the development process are supported by specific development tools to design, simulate, implement, analyze, verify and test the system. For example, DOORS[^doors] is used for managing requirements, EnterpriseArchitect[^ea] for UML design, the WindRiver Workbench[^workbench] as an editor and compiler for implementation and FaultTree+[^faulttree+] for safety and reliability analysis.

Discipline-specific development tools are also used – control engineers might use MATLAB/Simulink[^matlab], mechanical engineers might use Solidworks[^solidworks] for computer-aided-design (CAD), electrical engineers use Xilinx[^xilinx] for logic design and software engineers use a C compiler[^gcc] for software development. Besides dedicated development tools, even word processing applications and spreadsheet applications are used as development tools. In addition, there are tools for project management, collaboration, configuration management and product lifecycle management.

[^doors]: http://www.ibm.com/software/awdtools/doors
[^ea]: http://www.sparxsystems.com/products/ea
[^workbench]: http://www.windriver.com/products/workbench
[^faulttree+]: http://www.isograph-software.com/ftpover.htm
[^matlab]: http://www.mathworks.com/products/simulink
[^solidworks]: http://www.solidworks.com
[^xilinx]: http://www.xilinx.com/ise
[^gcc]: such as http://gcc.gnu.org
1.2. PROBLEM FORMULATION

The tools for embedded systems development are typically commercial off the shelf tools (COTS), but open source tools and in-house developed tools are used as well. A brief overview of commonly used development tools for embedded systems development can be found in [69]. In addition to the diversity, a large number of development tools are used in the embedded systems industry – the Automotive Technology Business Sector of Robert Bosch GmbH reports a portfolio of approximately 1,500 different tools [84].

1.2 Problem Formulation

In this section, the basic problem of tool integration is described (Section 1.2.1), typical stakeholders of tools and tool chains are analyzed (Section 1.2.2), and the challenges of the development of tool chains are identified (Section 1.2.3).

1.2.1 Basic Problem of Tool Integration

The tool landscape for embedded systems is both large and diverse, consisting of a number of specialized tools that support specific aspects of embedded systems development. Each tool focuses on supporting particular development tasks or aspects of the development process. Development tasks are not performed in isolation, but as parts of a development process. For a comprehensive tool support throughout the embedded systems development process, an engineering environment that supports the system development process by allowing for the seamless use of several development tools is required. Such an engineering environment needs to support sharing and reusing artifacts created with different tools, creating traces between these artifacts, invoking services of several tools and notifying developers. The data and services of the tools are not readily accessible due to the technical differences among the tools and are difficult to use due to the syntactic and semantic differences. While the technical and syntactic differences among the tools can be regarded as contributors to the accidental complexity of tool integration, the semantic differences contribute to the essential complexity (see [50] for a discussion of essential and accidental complexity). No ready-made solutions for such engineering environments are available due to the diversity and size of the landscape for development tools.

Tool integration is the activity of producing an engineering environment of tools that supports the systems development process [163]. Characteristically for this situation, the tools are available first, are built without considering integration, and their integration is performed afterwards. This situation is also described as a-posteriori integration [51], in contrast to a hypothetical a-priori integration, where tools are built so they fulfill specific integration requirements. Some tools may provide limited capabilities for a-priori integration with a limited set of tools, but in general, the need for a-posteriori integration can be assumed, since mostly commercial off the shelf (COTS) development tools are used [53].
CHAPTER 1. INTRODUCTION

In its simplest form, the engineering environment may be realized as a selection of tools and a set of guidelines that users of the tools, i.e. the embedded systems developers, need to follow to realize the integration. In this case, integration is performed by human operators, most likely the embedded systems developers themselves. This situation results in inefficiencies and high costs of development [164].

More effective engineering environments, called tool chains, are realized as software solution. Tool chains are effective, since integration-related activities are automated. This allows embedded systems developers to focus on the creation of the product instead of manually performing the integration. Tool chains are not necessarily connected in a linear way, but can form networks of tools. Tool chains can enable higher reuse of development artifacts across tools and they enable the automation of development activities. The assumption of the research community is that the main benefits of using tool chains are increased productivity [164], cost savings, reduced human error and reduced time to market [168].

1.2.2 Stakeholders

Here, some of the most important stakeholders of tools and tool chains are introduced, including their roles as users or creators of tools and tool chains. The embedded systems developers work with multiple development tools and take on the role of the users of tools. In addition, the embedded systems developers take on the role of the users of tool chains, motivated by the expected efficiency gains in development provided by tool chains. The vendors of tools for embedded systems development take on the role of the creators of tools.

Based on observations and interactions with industry in the research projects iFEST [2] and CESAR [1], the assignment of the role of the creator of tool chains is not clearly defined in industry. The role might be assigned to third party integration developers, but also to embedded systems developers or to tool vendors, which is problematic. Tool vendors are mostly interested in connecting only their tools to other tools, resulting in a limited scope of the integration, so called “islands of integration”. For embedded systems developers, the implementation of a tool chain is an additional burden that distracts them from their primary task of developing an embedded system.

1.2.3 Challenges of Tool Chain Development

A tool chain, is a software solution to the tool integration problem. The development of tool chains deals with a number of challenges.

Tailoring

In general, a tool chain supports a development process by automating the integration-related tasks of that process. Traditional tool chains support simple connections
1.2. PROBLEM FORMULATION

between a small number of tools, such as the connections between editor, compiler and linker. Tool chains with a simple architecture can thus be realized by a pipe and filter architecture [148]. Modern tool chains need to support development processes that are model-based and iterative [157] and include a larger number of tools used in different phases of the system lifecycle.

To get a tool chain accepted it is important that the users are involved [59] and the tool chain is customized to their individual requirements. Each development project for an embedded system may use a specific selection of development tools, and a specific development process. Due to the large number of development tools (see Section 1.1.1) and even larger number of possible tool combinations, one-size-fits-all tool chains cannot provide adequate support. Instead, tool chains need to be tailored to the product development process, development team and development tools of each development project. As a consequence, a number of different, tailored tool chains need to be developed.

Methodology

An established methodology for the development of tool chains is not available. Tool chains are often implemented ad-hoc, leading to “fragile integrations” [67] that are difficult to extend and maintain. There is insufficient support for synthesis and early analysis of tool chains – current platforms for tool integration provide partial solutions, but also introduce accidental complexity through the amount of coding, low-level technologies and configurations. Tool chains are manually implemented and can only be tested after the time and effort for implementation has been spent.

Efficient Development

Despite recent innovations in the field of tool integration, for example by tool integration frameworks, such as Jazz [78] or ModelBus [85], the development of tool chains still requires a large, manual implementation effort. In addition, tool integration frameworks require a number of configuration files with complex dependencies between them. Techniques are needed to improve the efficiency of building tool chains.

Description at an Appropriate Level of Abstraction

When working with industrial partners on tool integration, a gap between existing general purpose modeling languages and concepts from the domain of tool integration quickly became apparent. Integration approaches typically focus on describing the data of tools in the form of models and corresponding metamodels; the tool chain, however, is not explicitly modeled using concepts from the domain of tool integration. Instead, tool chains are built in a bottom-up manner and are often described on implementation level, using source code and the concepts offered by integration frameworks [78, 85]. These concepts are very detailed and depend on the use of certain technology.
Alternatively, tool chains are described by existing languages, such as general purpose modeling languages and languages for other domains. Borrowing descriptions and languages originally defined for other domains is a “workaround”, which introduces accidental complexity [14], an example is the use of business process models for describing tool chains [85].

Managing Tool Integration Views and Technologies

Several specialized technologies for realizing parts of tool chains are currently available, such as model transformation tools, tracing tools or libraries for exposing services of tools. Each of these integration technologies describes one aspect of the tool chain – a complete tool chain needs to cover several aspects. The functionality of a tool chain is thus fragmented across several integration technologies realizing parts of the tool chain. Each integration technology provides different, partial views of the tool chain, which are described using specific languages or configurations. To obtain an overview of the actually deployed or required functionality of the complete tool chain, several partial descriptions have to be consulted. The relationship between these descriptions is neither explicitly described nor managed, so inconsistencies can occur. This situation contributes to the accidental complexity of tool chain development.

Integration Conventions

Ad-hoc realizations of tool chains may use a variety of data formats, communication protocols and assumptions, which leads to incompatible tool chain realizations. Integration conventions provide a common ground for building tool chains and increase the likelihood that parts of tool chains can be reused in a different context than they were originally designed for, even when they are provided by a third party. Several conventions for integration exist, such as XMI (XML Metadata Interchange) [133], OSLC (Open Services for Lifecycle Collaboration) [141] and STEP [92]. Tool chains should be built according to conventions that are in use and accepted by industry. However, creating tool chains that follow integration conventions, involves a development overhead.

Cost-efficient Tool Chains

Industrially applied tool chains exist in a business context and need to be economically motivated, a view which has not received much attention in the academic literature on tool chains [164]. Tool chains are introduced by industry if they have a positive net value, i.e. if their benefits outweigh the necessary investment. The potential benefits of tool chains are increased productivity [164], cost savings, reduced human error and reduced time to market [168]. The necessary investment includes, for example, the cost for the construction of tool chains, the cost for introduction of the tool chain and training costs. One approach to increase the net value of tool chains is to develop them as cost-efficiently as possible.
1.3 High-Level Goals

The main goal of this work is to provide a language, methods and tools to describe and develop tool chains. It requires overcoming the challenges identified in Section 1.2.3. The main goal can be broken down into the following sub-goals:

1. **Support for describing tool chains in a systematic and structured way:** The building blocks of tool chains and their composition need to be identified and described at an appropriate level of abstraction. Such a description can help to manage different tool integration views and technologies.

2. **Support for developing tool chains efficiently:** Tool chains need to be efficiently realized as software solutions that are compliant with integration conventions.

To guide the work towards realizing these goals, a number of research questions are put forward (Section 2). For each of these goals, also a number of concrete criteria are identified (Section 3) using the **Goal Question Metric (GQM)** [18] approach. These criteria will be used for evaluating at the end of the work, whether the proposed results fulfill the stated goals.

1.4 Delimitations and Alternatives

A number of explicit delimitations are made for this work.

- An investigation of the implications of product lifecycle management and software configuration management in the context of tool chains is part of future work.

- The integration demands for co-simulation with multiple solvers are not addressed in this work. A description of the challenges and a possible solution for the co-simulation problem with multiple solvers is provided by the MOD-ELISAR\(^9\) project.

- The research results are internally validated based on case studies provided by industrial partners in the research projects iFEST [2] and CESAR [1]. Large-scale, external validation by industry is part of future work.

- Tooling for embedded systems development could be approached at a deeper level by building new tools [67] that provide an a-priori integration and thus render the a-posteriori tool integration problem irrelevant [51]. The challenges for constructing tool chains are reduced (Section 1.2.3), but the basic challenges of multi-view modeling remain (Section 1.2.1). In contrast, a pragmatic perspective of embedded systems development is taken, focusing on existing COTS development tools and the improvements that can be made by their integration.

CHAPTER 1. INTRODUCTION

1.5 Outline

This work is organized in four parts. Part I describes the context of this work. In Chapter 2, the research questions and the methods applied in this work are presented, structured according to two methodological research frameworks. In Chapter 3, a number of criteria are identified for characterizing the description of tool chains and their development. In Chapter 4, a brief overview of the state of the art in tool integration is provided, including a characterization of selected approaches with the previously identified criteria.

Part II presents an overview of the results and their evaluation. In Chapter 5, the design of the Tool Integration Language (TIL) is described. In Chapter 6, semi-automated techniques for different tasks of the tool chain development process are presented. In Chapter 7, the results are evaluated using the previously identified criteria.

In Part III, the results are discussed and conclusions are drawn. New perspectives provided by the work are discussed in Chapter 8. In Chapter 9, the main results are related to the research questions and an outlook for future research in the field is presented.

In the final Part IV, papers containing detailed descriptions of the results and evaluations are appended. Initial ideas for the language are described in Appended Papers A and B, and the language is described and evaluated in Appended Paper C, as visualized in Figure 1.1. The support for tool chain development is described in Appended Papers C - I, as visualized in Figure 1.2. Appended Paper E describes the support for specifying a tool chain that is aligned to a development process, Appended Papers F and G describe support for refining a tool chain model, Appended Papers H and I describe methods for analysis of a tool chain model and Appended Papers C and D describe code generation from a tool chain model.
Figure 1.1: Overview of Appended Papers A - C, which present the systematic description of tool chains with a language. The picture also documents the evolution of the language.
Figure 1.2: Overview of Appended Papers C - I, which present development support for tool chains. The picture shows the tool chain development process described in SPEM [135] notation and highlights the provided development support.
Chapter 2

Research Questions and Methods

In this chapter, the research questions that guided this work are introduced, their selection is motivated, and the methods applied to answer them are presented. Two methodological frameworks are introduced (Section 2.1) and applied to present the research questions and methods (Section 2.2).

2.1 Methodological Frameworks

Methodological frameworks provide a means for structuring and analyzing a chosen research approach. In this work, research on engineering design and software engineering is performed, thus both Blessing’s framework [45] for engineering design research (Section 2.1.1) and Shaw’s framework [147] for software engineering research (Section 2.1.2) are introduced.

2.1.1 Methodological Framework for Engineering Design Research

Engineering design research has two objectives, the development of design theories and the development of tool support [45]. Blessing proposes the Design Research Methodology (DRM) as a framework, which provides methods and guidelines for performing design research in the following stages:

- **Research Clarification:** to identify the topics of relevance and contribution, clarify the current understanding, clarify the research questions and create a research plan.

- **Descriptive Study 1:** to gain an in-depth understanding of the existing situation, e.g. by a literature survey.

- **Prescriptive Study:** to create support for improving the existing situation.
• Descriptive Study 2: to evaluate the support developed in the prescriptive study.

2.1.2 Methodological Framework for Software Engineering Research

In general, software engineering seeks better ways to develop and evaluate software, motivated by practical problems, key objectives typically being quality, cost and timeliness. Shaw [147] proposes a methodological framework for research in software engineering. The framework provides a classification scheme for characterizing a research approach in terms of research questions, results and validation.

• Research Question: To establish what is interesting, research questions are put forward. Typical research questions in software engineering can be classified as methods of development, methods for analysis, design and evaluation of a particular software, 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 as 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 (language and tool support for this language), specific solutions, judgments and reports.

• Research Validation: Validation can be accomplished through experience with the newly developed program or through systematic analysis [147]. The list of validation techniques used in software engineering includes analysis, experience, toy example, slice of life and evaluation.

2.2 Research Questions and Methods

The research questions of this work are motivated by several initial case studies [7, 8, 32] of tool chain development for embedded systems. The initial case studies used state-of-the-art tool integration technologies, such as model transformations and tool integration frameworks. This exposed the challenges of developing tool chains (see Section 1.2.3) and lead to the main research question addressed in this work:

• RQ1: How can the development of tool chains be systematized and automated?

To be able to phrase the main research question more sharply, the hypothesis of domain-specific languages is applied as an auxiliary hypothesis, which can be phrased as AUX-HYP: A domain-specific language offers appropriate constructs
and notations from the domain and is the basis for automated development support [123]. With the auxiliary hypothesis, the hypothesis of this work on tool chain development can be phrased as

- **HYP**: The development of tool chains can be supported by a systematic description of tool chains and by automated development methods based on the description.

With this hypothesis, RQ1 can be refined into RQ2 and RQ3.

- **RQ2**: How can tool chains be described systematically?

The results for RQ2 are presented in Chapter 5. Based on an initial case study [32], a modular approach for data-integration in tool chains is proposed (Appended Paper A), identifying two separate stages of data-transfer. Based on an additional case study with a tool integration framework [7, 8], the modularization is extended (Appended Paper B). A domain analysis [71] of tool chains is performed to identify common building blocks, which are organized into a modeling language. This modeling language can be applied to express a tool chain from an industrial research project (Appended Paper C). After evaluating the *descriptive* use of the modeling language, we explore its *constructive* use for supporting various development tasks.

- **RQ3**: To what extent can the development of tool chains be supported by automated methods?

Since the development of tool chains covers several development phases, we refine RQ3 into more refined research questions addressing automated methods for specific phases in the tool chain development process, such as specification, refinement, analysis and synthesis. The refined research questions and the results for RQ3 are presented in Chapter 6. Support for the specification of tool chains is proposed (Appended Paper E) and support for the refinement of tool chains (Appended Papers F and G). In addition, support for the analysis of tool chains is proposed (Appended Papers H and I) and the synthesis of tool chains (Appended Papers C and D).

### 2.2.1 Application of the Methodological Frameworks

Applying the methodological framework for engineering design research (see Section 2.1.1) shows that the research was performed in three cycles, as depicted in Figure 2.1. Each cycle started with a research clarification and descriptive work to enhance the understanding of the state of the art through literature studies and case studies, followed by prescriptive work for describing new methods and tools and another round of descriptive work to evaluate the feasibility and usefulness of the proposed methods using case studies to extract the lessons learned. The case studies are used to evaluate proposed approaches, but also to gain insights into the problem, which are used as a starting point for the next cycle of research. Each
cycle builds on the results and insights from the previous cycle. The first cycle in Figure 2.1 represents the development of a specific instance of a tool chain and is described in the Licentiate Thesis [27]. The second cycle describes a generalization of previous results, by proposing a language that can be used to describe tool chains. The results are described in Chapter 5. The third cycle explores possibilities for automating parts of the tool chain development process based on the developed language. The results are described in Chapter 6.

Applying the methodological framework for software engineering research (see Section 2.1.2) yields the classification presented in Table 2.1. For each tackled research question, the type of research question, the type of research result and the type of research validation are identified. The research questions address methods of development and characterization schemes. The results are descriptive models, methods and tools. The results are validated both qualitatively and quantitatively by case studies that were provided by industrial partners in the iFEST [2] and the CESAR [1] project in the form of textual descriptions. Due to their practical relevance, they can be classified as a slice of life according to Shaw’s terminology [147]. Our choice of case studies as validation method is in line with the statistics on research in software engineering presented by Shaw [147], where validation is typically performed by examples, when the research questions are classified as development methods, and research results are classified as notation and tool.

Table 2.1: Classification of research questions, results and validation

<table>
<thead>
<tr>
<th>RQ</th>
<th>Type of Research Question</th>
<th>Type of Result</th>
<th>Type of Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ1</td>
<td>Method of Development</td>
<td>Method and Tool</td>
<td>Slice of Life</td>
</tr>
<tr>
<td>RQ2</td>
<td>Characterization Scheme</td>
<td>Descriptive Model</td>
<td>Slice of Life</td>
</tr>
<tr>
<td>RQ3</td>
<td>Method of Development</td>
<td>Method and Tool</td>
<td>Slice of Life</td>
</tr>
</tbody>
</table>
2.2. RESEARCH QUESTIONS AND METHODS

2.2.2 Case Studies

The following case studies are used for the validation of different parts of the approach:

1. As part of the CESAR project [1], a tool chain of MATLAB/Simulink, PapyrusUML and the HiP-HOPS tool has been created with representatives from the automotive industry. For this case study a previous implementation was carried out using state-of-the-art integration technology [7, 8]. Executing the case study with TIL, provides opportunities for comparison. This case study is used for evaluating several methods supporting the development of tool chains, including code generation.

2. As part of the iFEST project [2], a case study is provided by our industrial partner, who develops embedded systems characterized by tightly coupled hardware and software components: The application logic is partitioned into software running on a Freescale PowerPC and hardware logic realized by a Xilinx FPGA. The scenario requires a number of tools for modeling, simulation, analysis, code-generation and coding. Today, no tool chain is in place to support the development process. The vision of an industrial tool chain for hardware-software co-design is described textually and in the form of sketches. This case study is used for evaluating several methods supporting the development of tool chains.

3. SPEM process models from the AUTOSAR (Automotive Open Software Architecture) [15] specification defined by the automotive industry are used as a case study for the construction of a tool chain that is aligned to a process model. This case study is selected, because of the industrial relevance of AUTOSAR specification and the public accessibility of the process models in the specification.

Case studies are applied in a specific context, and thus the possibility to generalize the obtained results requires particular discussion. Case studies have limitations regarding their generalizability, which can be controlled to the effect that the ”generalizability of case studies can be increased by the strategic selection of cases” [76]. The literature [145, 146] differentiates between representative and extreme case studies. To cover a wide range of cases and to increase the generalizability, both representative and extreme cases need to be chosen for the evaluation.

Case study 2 reflects the tool chain of an industrial partner in the iFEST project. It is representative for the tool chains of other industrial partners studied in the iFEST project regarding the following parameters:

- **Number of tools**: The tool chains in the iFEST project contain less than ten tools.

- **Types of tools**: A mix of mostly COTS tools and in-house developed tools are used. COTS tools typically include MATLAB/Simulink and UML tools.
• **Size of the exposed data and services of the tools**: The size of roughly 10 entities with each up to 10 attributes. The templates for the exposed data and services proposed by the integration convention OSLC (Open Services for Lifecycle Collaboration) [141] have a similar or smaller size.

• **Different aspects of integration**: The integration aspects data-transfer, data-tracing, control-invocation, control-notification and process integration are considered.

• **Types of users in the tool chain**: Four user roles are defined, representing different engineering disciplines: a systems engineer, control engineer, software engineer and hardware engineer. The disciplines are typical for embedded systems engineering (see Section 1.1).

Apart from representative cases, also extreme cases are studied, since the latter can reveal more information and increase the generalizability of the case study [76]. The case studies presented in Appended Paper D focus on extreme values of a critical part of the tool chain, namely the size of the exposed data and services of the tools. The behavior of the generator is explored for a metamodel (the complete metamodel of UML\(^1\)) that has over 20 times the typical size.

\(^1\)\url{http://www.eclipse.org/uml2}
Chapter 3

Criteria for the Description and Development of Tool Chains

The research questions introduced in the previous section provide the general direction towards the high-level goals stated in Section 1.3. To make these goals more concrete and measurable, a number of criteria for characterizing the description and development of tool chains are identified in this section. For selecting the criteria in a structured way, the Goal Question Metric (GQM) approach [18] is applied: starting from high level goals, a number of questions are defined that make these goals operational, and a number of metrics for each question that make the answers measurable. For simplicity, the questions and associated metrics are called criteria in the following. These criteria are used in later sections for characterizing related work and for evaluating the research results. In this work, the two main high-level goals address the description and development of tool chains.

The first goal is to provide an appropriate description of a tool chain; this goal is related to RQ2. In Section 3.1, a number of questions are elicited to cover several viewpoints for the description of tool chains, including appropriate metrics for measurement of the respective questions.

The second goal is to provide development support for tool chains; this goal is related to RQ3. In Section 3.2, a number of questions are elicited to capture the extent of the provided support for the specification, analysis and synthesis of tool chains. The metrics measure if appropriate support is provided and the degree to which the support is automated.

The criteria presented in this section originate from various sources: the analysis of the literature, earlier case studies and interactions with industrial stakeholders. The lessons learned have been gathered from earlier case studies with manually programmed tool chains built with model transformations [32, 39] and with a state of the art tool integration framework [7, 8]. In addition, the research project iFEST [2] provided a forum to work with different types of stakeholders in tool chains: potential industrial users of tool chains, vendors of development tools and vendors
CHAPTER 3. CRITERIA FOR THE DESCRIPTION AND DEVELOPMENT OF TOOL CHAINS

20

of integration platforms.

3.1 Criteria for the Description of Tool Chains

Tool chains can be described from various viewpoints, leading to views addressing different concerns and aspects of tool chains. In addition, an overview of the tool chain can be described. The description can be domain-specific to tool integration and have a graphical representation to support communication among stakeholders. Each of these criteria will be further detailed below.

3.1.1 Coverage of Integration Concerns

The description of a tool chain may address different integration concerns, which can be classified as follows:

- The technical concern describes the technical means to access the data and services of a tool. Examples for describing technical concerns are source code, data files, communication protocols or APIs (Application Programming Interfaces) of the tools in a tool chain.

- The structural concern describes the structure of the data and the signature of functions exposed by the tools in a tool chain. Examples of approaches describing structural concerns are metamodels or grammars.

- The semantic concern describes the meaning of data and services of different tools in a tool chain. Examples of approaches describing semantic concerns are semantics-preserving model transformations or ontologies.

- The architectural concern describes the “fundamental concepts or properties of a tool chain embodied in its elements and relationships” [93]. An approach describing the architectural concern of a tool chain may describe tools as components and the integration between tools as connectors.

To build a tool chain, usually all concerns need to be addressed, however, not all concerns are explicitly described. To describe a tool chain, the characteristics of integrated tools need to be described, which are covered by technical and structural concerns, and the characteristics of sets of integrated tools, which are covered by semantic and architectural concerns. In Figure 3.1 the integration concerns are ordered, so the scope is decreasing and the amount of details is increasing (from top to bottom).

This criterion addresses the description of the concerns, in contrast to related classifications, which address the features of interoperability. The features of interoperability can be classified, for example by the Levels of Conceptual Interoperability Model (LCIM) used in the simulation community [162].

This criterion addresses the question: Which integration concerns are covered by the description of the tool chain? The criterion is measured in terms of the set
3.1. CRITERIA FOR THE DESCRIPTION OF TOOL CHAINS

Figure 3.1: Tool integration concerns

of covered concerns: technical concern, structural concern, semantic concern and architectural concern.

3.1.2 Coverage of Integration Aspects

Wasserman introduced several integration dimensions to delimit the scope of tool integration [163]: data integration, control integration, process integration, platform integration and presentation integration. This classification is refined by splitting both data integration and control integration into two classes to provide finer distinctions. The resulting classes are called integration aspects.

This criterion addresses the question: Which integration aspects are explicitly described at an appropriate level of abstraction? For measurement of this criterion, we use a set of the values data-transfer, data-tracing, control-invocation, control-notification, process, platform and presentation. The different aspects of tool integration are visualized in Figure 3.2 and explained in the following.

- Data-Transfer Integration Aspect: Data integration can be realized by transferring data between the integrated tools. In general, the integrated tools expect the data according to different structures, which requires conversion of the tool data in a semantics-preserving manner.

- Data-Tracing Integration Aspect: Data integration can also be realized by linking data between the integrated tools by traces. The linking of fine-grained data elements of the tools requires support for creation and management of links. In this context it is important to distinguish between the description of trace instances and the description of the possibility to create traces between certain types of data.
CHAPTER 3. CRITERIA FOR THE DESCRIPTION AND DEVELOPMENT OF TOOL CHAINS

22

Figure 3.2: The tool integration aspects

- Control-Invocation Integration Aspect: Control-invocation allows users or parts of the tool chain to invoke services of other tools or parts of the tool chain.

- Control-Notification Integration Aspect: Control-notification allows tools or parts of the tool chain to pass information to the tool chain user.

- Process Integration Aspect: Process integration allows the tools of the tool chain to interact according to a description of the development process. To realize process integration, the tool chain needs to be aligned to the development process. In addition, the tool chain needs to have a notion of the state of the development process, which can be either realized dynamically, using events, or statically, using a predefined sequence. Tool chains with event-triggered process integration stay in the background monitoring events that trigger tool chain activity. The events can be caused by users or by other tool chain activities. Tool chains with sequence-based process integration perform a number of activities in sequence, with little user interaction. The need for alignment between the tool chain and the development process, requires the possibility to describe how the tool chain reacts to certain events in the development process.

- Platform-Distribution Integration Aspect: In globally distributed product development, specific phases of the development process might be handled at remote locations, thus the description of the tool chain needs to allow for describing the tool chain as a distributed system. To enable a smooth, distributed development process, the tools from different locations need to be integrated into a globally distributed tool chain [87]. Here the explicit description of platform integration is of interest.

- Presentation Integration Aspect: Presentation integration provides a common user interface with a unified look and feel for the data and services of the tools
in the tool chain.

### 3.1.3 Overview of the Tool Chain

As tool chains become larger in size and span multiple integration aspects, an overview of the tool chain is important to take all relevant integration aspects into account. This criterion addresses the question: *Can the description provide a coherent and complete overview of the whole tool chain including all integration aspects?* The criterion is measured in terms of the values support and missing support for an overview of the tool chain.

### 3.1.4 Domain-Specific Description

Describing systems using general purpose languages or languages created for different domains, may introduce accidental complexity, i.e., make the description of the system more complex than necessary. This criterion addresses the question: *Is the tool chain described using specific concepts from the domain of tool integration?* For measurement of this criterion, a distinction is made between a description that uses concepts from the *domain of tool integration*, a description that uses generic concepts or borrows concepts from *other domains*. The use of source code is not considered.

### 3.1.5 Graphical Representation to Support Communication

Problems in interdisciplinary communication often arise from the lack of a shared terminology [4]. A boundary object is defined as “an analytic concept of those scientific objects, which both inhabit several intersecting social worlds and satisfy the informational requirements of both of them” [149] and can thus be a means to support interdisciplinary communication. Here the tool chain description is an object in the world of the various stakeholders of tool chains (see Section 1.2.2). A description of the tool chain that the various stakeholders can relate to, can become a *boundary object* that may facilitate communication.

The tool chain description should allow communication of an envisioned or already existing tool chain to different stakeholders, and should have a *graphical representation*. Furthermore, the graphical representation should be consistent with the implementation of the tool chain. This criterion addresses the question: *Which parts of the tool chain can be graphically represented by the description?* For measurement of this criterion, the values *whole* tool chain or *parts* of the tool chain are used to describe the scope of the graphical representation.

### 3.2 Criteria for the Development of Tool Chains

As described in Section 1.2, there is a need for methods and tools for supporting the development of tool chains. The goals of this support are to provide a structure
for the development process and to reduce the overall effort required for developing tool chains. Methods and tools should support the various activities of tool chain development. In this work we focus on the following activities:

- **Specification:** The description should allow specification of a tool chain and capture the important design decisions involved in creating a tool chain. Specification methods should support the creation and refinement of the description of a tool chain.

- **Analysis:** The description of a tool chain should be analyzable to check and estimate the properties of the tool chain. Analysis methods should be available to analyze both functional and non-functional properties of the tool chain. Analysis methods can be used to determine if the described tool chain can fulfill the requirements. For an envisioned tool chain, the analysis should be performed early in the development process, before the tool chain is implemented. For an existing tool chain, the analysis can provide guidance for improvements of the design.

- **Synthesis:** The description of a tool chain should be a blueprint for the realization of the tool chain. Synthesis methods should be available to create an implementation of the complete tool chain, which should be consistent with the description of the tool chain.

Besides the above activities, which are in the scope of this work, additional activities could be supported. Support could be provided by methods for validating the tool chain against the expectations of the users, methods for testing the implementation of the tool chain, methods for the evolution and maintenance of tool chains and methods for the deployment of tool chains. We leave the criteria for evaluating the support for these development activities to future work.

### 3.2.1 Specification Support: Tool Chain

A tool chain needs to be tailored to the set of development tools and the product development process that it is intended to support. This criterion addresses the question: *To what extent is the creation of an overall specification of the tool chain supported?* It is measured in terms of the values *automated, semi-automated* and *manual* and *unavailable* specification support.

### 3.2.2 Specification Support: Tool

The data and services of the tools that are integrated by the tool chain need to be specified. The criterion addresses the question: *To what extent is the activity of specifying the integrated tools supported?* The extent of automation is measured in terms of the values *automated, semi-automated, manual* and *unavailable* specification support.
3.2. CRITERIA FOR THE DEVELOPMENT OF TOOL CHAINS

3.2.3 Specification Support: Other Tool Chain Parts

Besides the tools, other parts of the tool chain, such as the data-transfer, invocation, notification and tracing, also need to be specified. The criterion addresses the question: *To what extent is the activity of creating the specification of the remaining tool chain parts supported?* It is measured in terms of the values *automated, semi-automated, manual* and *unavailable* specification support.

3.2.4 Analysis Support: Verification

The description of the tool chain needs to be correct and it needs to fulfill the given requirements. This criterion addresses the question: *To what extent is the creator of a tool chain supported by methods that check that the requirements are met by the tool chain description and that the tool chain description is correct?* This criterion is measured in terms of the values *automated, semi-automated, manual* and *unavailable* verification support.

3.2.5 Analysis Support: Non-Functional Properties

Besides the functional correctness of the tool chain, the non-functional properties (NFPs) of tool chains need to be within acceptable limits. For the largest possible leverage, the analysis should be performed before the tool chain has been built. This criterion addresses the question: *To what extent is the creator of a tool chain supported in analyzing the non-functional properties of a tool chain, such as the construction cost of the tool chain?* This criterion is measured in terms of the values *automated, semi-automated, manual* and *unavailable* support for analysis of NFPs.

3.2.6 Synthesis Support: Tool Chain

The tool chain realization consists of a number of parts that realize the connections between the tools. This criterion addresses the question: *To what extent can the implementation of the complete tool chain be synthesized from the specification?* This criterion is measured in terms of the values *automated, semi-automated, manual* and *unavailable* support for synthesis.

3.2.7 Synthesis Support: Tool

To realize the tool chain, wrappers for each of the integrated tools are created to expose the data and services of the tools. This criterion addresses the question: *To what extent can the wrappers for the integrated tools be synthesized from their specification?* This criterion is measured in terms of the values *automated, semi-automated, manual* and *unavailable* support for synthesis.
CHAPTER 3. CRITERIA FOR THE DESCRIPTION AND DEVELOPMENT OF TOOL CHAINS

3.3 Summary

The goal of this work is to devise methods and tools for the description and development of tool chains. A number of criteria to characterize the description and development of tool chains were selected and consist of a question and a metric. The criteria and will be used to evaluate both existing approaches (Chapter 4) and the results of this work (Chapter 7).
Chapter 4

State of the Art

In this chapter, an overview of the state of the art in tool integration is provided. First, some relevant basic concepts (Section 4.1) are introduced. Then, state of the art tool integration approaches are presented (Section 4.2) with a focus on model-based tool integration approaches (Section 4.2.1) and service-oriented tool integration approaches (Section 4.2.2). Finally, selected state of the art approaches are characterized according to the previously elicited criteria (Section 4.3).

4.1 Basic Concepts and Terminology

In this section, some basic concepts relevant to the area of tool integration are introduced. The concepts are from the area of models, languages and architecture and are relevant since they are used throughout the thesis.

Model-based and model-driven approaches: use explicitly represented models as primary artifacts for describing a system [131]. Model-based approaches focus on models as important artifacts for describing a system [156]. Model-driven approaches are a specialization of model-based approaches, where models are not just used descriptively, but they are used as construction tools [156]. During development, a series of such models are specified, refined and transformed. 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: “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” [24]. Several kinds of models exist [156], such as mental models, conceptual models, formal models and executable models.

Metamodelling Levels M0-M3: are defined by the OMG [132]. They describe the conformance relation between objects of the real world (M0), models (M1),
metamodels (M2) and metametamodels (M3), as depicted in Figure 4.1: a model conforms to a metamodel and a metamodel conforms to a metametamodel. Some metametamodels can be used for their own definition, a concept called meta-circularity [80]. (For a discussion on the definition of a metamodel, which is in conflict with the use of the Metamodeling Levels by the OMG, see [108].)

Figure 4.1: The metalevels of models and model transformations [27]

Model Transformation: is the “automatic generation of one or multiple target models from one or multiple source models, according to a transformation description” [122]. The model transformation description is expressed in a model transformation language, such as QVT (Query/View/Transformations) [137] or ATL (ATLAS Transformation Language) [96]. 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 conforms to a model transformation metamodel (see Figure 4.1). In [26] the state of the art in model transformation is reviewed, including the terminology, usage scenarios, a classification scheme of the problems that are solved by model transformations, a classification scheme of model transformation languages and an overview of several model transformation languages and engines.

Language: comprises “the set of abstract syntax models. No meaning is given to these models” [80].

Concrete Language: “comprises both the abstract syntax and a concrete syntax mapping function. A single language may have several concrete languages associated with it” [80].

Formalism: “consists of a language, a semantic domain and a semantic mapping function giving meaning to a model in the language” [80].
Concrete Formalism: “comprises a formalism together with a concrete syntax mapping function” [80].

Semantic Mapping Function: maps the abstract syntax of a language to a semantic domain and can be described formally or informally [57].

Domain-Specific Modeling Language: is a modeling language tailored to a specific application domain that captures the concepts of this domain and typically has a narrow scope [102, 123]. In contrast, a general purpose modeling language has a broad scope and is often standardized, an example is the Unified Modeling Language (UML) [139]. Customizations of a general purpose modeling language to capture the concepts of a specific domain, are a compromise between the above options. An example is the UML profile mechanism, a lightweight extension for UML [139].

Architecture: is defined as the “fundamental concepts or properties of a system in its environment embodied in its elements, relationships and in the principles of its design and evolution” [93].

Architecture Description: is a “work product used to express an architecture” [93] and typically comprises the architectural concepts configuration, component, connector and interface [153].

Architecture Description Language (ADL): is defined as “any form of expression for use in architecture descriptions” [93] and describe and formalize a high-level decomposition of a system. A number of surveys compare the ADLs with respect to their ability to describe generic architectures [60, 89, 119] or architectures of a specific domain. In that case, the ADL is capable of representing domain concepts, e.g. architectures of embedded systems [72]. An example is EAST-ADL [63], which is capable of representing concepts from the automotive embedded systems domain. Specialized ADLs for the tool integration domain are not available.

Standardized Exchange Format: Several standards and conventions for exchange formats have been proposed [104], but due to their large number, their generic applicability is limited. The XMI (XML Metadata Interchange) [133] format defined by the OMG (Object Management Group) is often used for serializing models. XMI is quite generic and does not specify the metamodels to be used for representing data for specific domains. More restrictive exchange formats specify the metamodels to be used for the data of certain domains, such as the STEP standard (ISO 10303) [92] and the OSLC (Open Services for Lifecycle Collaboration) [141] initiative. STEP defines several application protocols for specific domains, such as AP233 for systems engineering. Building OSLC-compliant tool chains requires the use of many technologies and is thus complex.
Ontology: is defined as a "formal, explicit specification of a shared conceptualization" [83]. This shared conceptualization can provide a common reference for several tools [126, 161] and each tool uses a view of the shared conceptualization [79]. Typically, a manual approach is used to define an ontology, but automated approaches exist as well [151]. Approaches either define a single, comprehensive ontology or a collection of several ontologies. When several ontologies are used, the links between the ontologies need to be defined separately. To reduce the effort of defining the mappings between ontologies, small domain ontologies are proposed [17].

4.2 Tool Integration

In the context of this work, we are concerned with the integration of development tools. The data that a development tool operates on is called tool data. The functionality or services that a development tool provides are called tool services. Tool integration is defined as the activity of producing an environment of tools that supports the development process by sharing and reusing artifacts created with different tools, tracing between these artifacts, invoking services of several tools and notifying developers (according to [163]). Tool integration is thus concerned with the relationships among tools, the properties of the relationship and the degree to which tools agree [154]. The outcome of tool integration is a seamless engineering environment of several development tools [52], and if it is a realized in software, it is called tool chain. The assumption of the research community is that a tool chain of several integrated tools increases productivity and product quality [164].

A broad overview of the literature on tool integration is provided in the annotated bibliographies of Brown [52] and Wicks [165]. The scope of tool integration is defined by the classification of integration dimensions [163], which is refined in Section 3.1.2 into the following integration aspects: data-transfer, data-tracing, control-invocation, control-notification, process, platform and presentation.

The technology, which is used to realize tool integration, is called integration technology. Typically, integration technology is generic technology that is applied in a specific way to the tool integration challenge. Often one technology may be used to realize a specific aspect of tool integration. For example, the data integration aspect may be realized using modeling technology, i.e. models, metamodels and model transformations (see Section 4.2.1) and the control and platform integration aspect may be realized using middleware technology and services (see Section 4.2.2).

The technical space (also called technological space) concerns the technology used for representation of the tool data [74, 109, 122]. Examples for technical spaces are EMF (Eclipse Modeling Framework) [150] or XML (Extensible Markup Language) [49]. Each technical space defines a format or protocol for representation and a set of technologies to work with the representation. The technical space for integration is the technical space used as common ground for integration of tools (see Figure 4.2). A commonly used approach for tool integration is a tool adapter,
4.2. TOOL INTEGRATION

which is software that provides the data and services of a tool in the technical space for integration (see Figure 4.2).

![Figure 4.2: Tools, tool adapters and the technical space for integration](image)

4.2.1 Model-Based Tool Integration

Model-based tool integration uses modeling technologies, such as models, metamodels and model transformations, to realize tool integration. Model-based tool integration focuses on the data integration aspect [100] and usually covers both data-transfer and data-tracing. The tool data is stored in tool-specific models, which correspond to tool-specific metamodels. Model-based tool integration provides means for specifying mappings between tool-specific metamodels to overcome the semantic, syntactic and technical heterogeneity between the tool data [100]. Model-based tool integration is primarily used for integrating modeling tools, but can also be applied to other types of tools that do not define an explicit metamodel. For such tools, a metamodel is ascribed to the tool allowing the data of the tool to be represented as a model. Almost all approaches use model transformations as the operational basis for data integration. Model transformations are either directly specified or generated from other representations. In the following, we present different building blocks for realizing model-based tool integration.

Model Transformations for Tool Integration: Model transformations are typically used as an executable representation for the mappings between metamodels, which describe the data of the tools [157]. Model transformations are either specified manually, generated from weaving models [65, 66] or generated from ontologies [90]. Several approaches explore the use of manually specified model transformations for tool integration, for example the MOFLON language [6] and Eclipse-based languages [70]. Model transformations can be applied for model synchronization [55, 81], to keep two or more models in a consistent state. A change in either of the models leads to an appropriate, incremental update of the other model(s).

Annotation-based Approaches: Annotation-based approaches allow the user to define annotations on the elements of one of the models (M1-level) involved
in the integration. The annotations are a support for realizing more accurate mappings, and are complimented with an executable mapping, usually in the form of model transformations. The model transformation rules use the information provided by the annotations and are relatively simple. For UML-based approaches, the annotations can be realized by stereotypes of the UML Profile mechanism. An example is the integration of SysML and Modelica [142]. The BIZYCLE approach uses annotations to prevent both structural and semantic inconsistencies [125].

Pivot Metamodel: If each tool had a connection to each other tool, the number of connections would be in $O(n^2)$, with $n$ being the number of tools. The pivot metamodel serves as a lingua franca, which reduces the number of connections between the tools to $O(n)$. Since each tool only needs to map to and from the pivot metamodel. The pivot metamodel is instantiated at run-time, and the resulting model is used as an intermediate model to represent data from several tools or modeling languages. Pivot metamodels are typically established for a specific domain, such as EAST-ADL2 [63] for the automotive embedded systems domain and the CESAR Common Metamodel [19] for the domain of safety-critical embedded systems. Pivot metamodels are also used for mappings between different development environments [54] and between architecture description languages with DUALITY [115], for visualization tools [114] and business processes [170]. The approach by Herzog [88] proposes to establish a common information metamodel for the exchange of systems engineering data.

Model Management Frameworks: Frameworks for model management 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 [22]. A model registry is provided for identifying, storing and retrieving both models and metamodels. An example of an early model management framework is Rondo [121]. It is increasingly recognized that not only the model, but also the mapping between different models needs to be managed by these platforms [23]. However, model management frameworks assume that the tool data is available in a shared technical space and thus typically do not cover the technical integration concern (see Section 3.1.1).

Model-Based Tool Integration Frameworks: Frameworks for tool integration provide reusable components for building tool chains. Model-based tool integration frameworks focus on the data integration aspect, and control integration is typically not present or a secondary issue. Examples are MOFLON [6], OTIF [101, 152], ModelBus [85] and ModelCVS [107]. The tool integration frameworks ToolNet [5] and TiE [105] are based on MOFLON, which provides MOF-based metamodeling and a model transformation engine. The Open Tool Integration Framework (OTIF) [101, 152] provides reusable com-
components and libraries to build tool chains and focuses on data integration. OTIF uses the GReAT model transformation language [16] for data integration, but control integration is out of scope. ModelBus [85] is a model-based and service-oriented integration framework and thus provides data and control integration. In addition it provides a repository for EMF models and support for ATL [96] and QVT [137] transformations. An example of a model-based tool integration approach using ontologies is the ModelCVS framework [97, 99, 107]. ModelCVS maps tool-specific metamodels to an ontology and relationships between the tool-specific ontology elements are defined.

4.2.2 Service-Oriented Tool Integration

In global engineering settings, engineers from different locations participate in the product development process [86]. Thus the development tools and activities are distributed over several locations [87] and as a result, tool chains become distributed systems that are realized on heterogeneous platforms. Service-oriented tool integration approaches use services to realize the control integration and platform integration aspects.

A wrapper for each development tool exposes the tool services. This wrapper is called a service-oriented tool adapter. Tool chains are realized by orchestrating the services provided by service-oriented tool adapters. In the following, we present different building blocks for realizing service-oriented tool integration.

Languages for Service Engineering: General purpose modeling languages such as UML can be applied for modeling general service-oriented solutions. Domain-specific modeling languages for services strive to expose the properties of services, resulting in a fine-granular description of services. Examples are the Service Oriented Modeling Framework [20] and approaches using UML profiles, such as the UML profile for software services [95], the SOAML UML profile for services [138] or the UML4SOA profile [82]. While these languages can be used for building tool chains, they provide a technology-centric description at a low-level of abstraction.

Languages for Workflows and Orchestration: Workflows are defined as “the organization of a structured application in an abstract fashion, such that the implementation of the atomic tasks being organized is independent from the organization itself” [117]. Workflow languages can thus be used for realizing process integration. A comparison of different workflow languages is provided in [159]. Orchestration is the centrally controlled interaction of services and is thus a means for expressing process integration. The orchestration of web services can for example be modeled with BPMN (Business Process Modeling Notation) [140], which is not executable, or with BPEL (Business Process Execution Language) [127], which can be executed. The Service Component Architecture (SCA) [128] is a middleware architecture for building service-oriented solutions and includes a language for describing the composition
of components. While these languages can be used for manually building service-oriented tool integration solutions, the languages do not provide specific support for tool integration.

Service-Oriented Tool Integration Frameworks: Jazz [78] is a service-oriented tool integration framework that provides storage and version management. The services implemented in Jazz use a RESTful (Representational State Transfer) [75] architecture and are compliant with the specification of the OSLC initiative (Open Services for Lifecycle Collaboration) [141, 166]. Another service-oriented tool integration framework is jETI [116], focuses on control, process and platform integration but data integration is not supported. It assumes that the integrated tools are command-line tools, thus tool chains are realized as sequences of tools. ModelBus [85] combines a model-based and service-oriented tool integration approach. It provides a repository for EMF models and support for ATL and QVT transformations. For each tool, a tool adapter is created that uses SOAP-based web services. These services can be orchestrated using a BPEL [127] engine for process integration. CORSET [68] is an emerging framework for the integration of web applications based on a domain-specific language. Since the integrated tools are web-applications, data and functionality are available in a common technical space for integration and no tool adapters are needed. The integration is realized using a browser-based mechanism to add a user interface and integration functionality to the integrated web applications. While the previous tool integration frameworks were generic, there are also integration frameworks for specialized application domains, such as the Kepler framework [113] for tool integration in scientific workflows and BiojETI [110] for remote tool integration for bioinformatics applications.

The OSLC initiative (Open Services for Lifecycle Collaboration) is an industrial effort that aims at standardizing different choices for tool integration solutions [141], such as the technical space for integration, the communication protocols and architecture for tool adapters, directory services, data representation, data structures and metadata. OSLC proposes a RESTful architecture [75] and the use of the use of the Linked Data Model [21]. An OSLC-compliant tool or tool adapter represents tool data in the form of an OSLC Resource, a protocol on top of the Resource Description Format (RDF) [106], and offers it for integration with other tools. A Resource Shape describes the data structure of the Resource and can be requested at run-time. An OSLC tool adapter furthermore consists of a ServiceProvider, ServiceCatalog, Resources and ResourceShapes. Resources represent the data of the tool and correspond to a model. ResourceShapes describe the structure of the Resource and are needed for reflection and discovery. The ServiceCatalog lists all the available services of the tool, such as data access methods. Each service in the ServiceCatalog is specified by a Query Capability to retrieve data and a Creation Factory to create new data. The necessary effort and cost for creating service-oriented
tool chains that are compliant to OSLC are considerable. The tool chains are complex service-oriented systems, which require web servers, configuration files, an implementation that conforms to the service-oriented standards and various metadata for the data and services. OSLC provides standardized communication protocols and data structures, but the heavy technological constraints contribute to the increasing complexity of tool chain development.

4.3 Characterization of Tool Integration Approaches

The following model-based and service-oriented tool integration frameworks are selected for a deeper analysis and characterization: ModelBus [85], OTIF [152], jETI [116], MOFLON [6], ModelCVS [98], OSLC [141] and CORSET [68]. These frameworks are analyzed according to the criteria for the description and development of tool chains that were introduced in Chapter 3.

4.3.1 Characterization of the Description of Tool Chains

The surveyed approaches provide different means for the description of tool chains. In the following the characteristics of these descriptions are analyzed according to the criteria introduced in Section 3.1.

Coverage of Integration Concerns

To analyze the coverage of integration concerns provided by the different approaches, the classification in Figure 3.1 is used. The descriptions used in the surveyed approaches cover the integration concerns to different extent.

All model-based integration approaches provide possibilities to describe semantic concerns by semantics-preserving model transformations and structural concerns by metamodels. MOFLON focuses on the description of semantic concerns by graph transformations and structural concerns are realized by MOF-conform metamodels. ModelCVS describes semantic concerns by an ontology for the integrated tool data and by the pattern-based model transformation language CAR. In addition to descriptions for structural and semantic integration concerns, OTIF covers the architectural integration concern by specifying two patterns for tool integration. The service-oriented approaches jETI and OSLC explicitly describes structural concerns by the definition of data structures. CORSET focuses on structural concerns by using syntactic analysis of web-pages that present tool data.

An outcome of the above analysis is that the technical and architectural integration concerns are often not described by dedicated languages. While the technical integration concern is typically described using source code, the architectural integration concern is not described at all by most approaches in the scope of this survey. OTIF is the only approach, which stresses the importance of the architecture for tool chains by suggesting two design patterns for tool integration that can provide conceptual guidance for creating the architecture of a tool chain. To use,
apply and combine these patterns for constructing a practically useful tool chain, additional support is needed.

**Coverage of Integration Aspects**

To analyze the coverage of integration aspects provided by the different approaches, the classification in Figure 3.2 is used.

- **Data-Transfer**: Model-based approaches, such as MOFLON, OTIF, ModelCVS and ModelBus, use a model transformation language or a pivot language to describe data-transfer. Service-oriented approaches such as jETI and OSLC do not offer specific descriptions for data-transfer.

- **Data-Tracing**: Model-based approaches, such as MOFLON and ModelBus, support traces as a byproduct of the execution of model transformations, however, tracing is not explicitly supported independently from the data-transfer realized by model transformations. Separate approaches for tracing need to be used, such as described in [124]. The service-oriented approach jETI does not offer specific descriptions for data-tracing, but OSLC provides concepts for tracing based on the linked data model.

- **Control-Invocation**: Service-oriented approaches, such as jETI, CORSET and ModelBus, can describe the invocation of tool services. Model-based approaches, such as MOFLON and ModelCVS, do not offer possibilities to describe invocation.

- **Control-Notification**: ModelBus is the only surveyed approach that allows for an explicit description of user notification. Other approaches may be able to realize user notification by implementation.

- **Process Integration**: Service-oriented approaches use either general-purpose workflow languages, languages for enacting business processes or executable process modeling languages. ModelBus uses BPEL [127] workflows, which were originally designed for automating business processes. ModelBus is able to describe sequence-based process integration, and defines a limited number of events on its repository, which allow for event-triggered invocation. jETI provides a sequence-based description of process integration, however, an events-based description is not available. The OSLC specification for automation offers an interface for sequence-based systems, but it needs to be described using source code. CORSET provides an event-triggered description of process integration.

- **Platform Integration**: Model-based tool integration approaches typically assume that all tools are deployed locally. Service-oriented tool integration approaches, such as jETI and ModelBus, support the integration of distributed tools.
Presentation Integration: OSLC is the only surveyed approach that offers a description for presentation integration by the concept of delegated user interfaces. Presentation integration is not explicitly described by the other surveyed approaches.

While most of the surveyed approaches are able to describe several integration aspects, no approach covers all integration aspects. For integration aspects that are not explicitly described by a specific approach, an implementation may still be created using source code.

Overview of the Tool Chain

The surveyed approaches cover the description of several integration aspects of tool chains and to express these integration aspects, separate descriptions are typically used: workflows are used to describe process integration (e.g. in jETI or ModelBus) and model transformation languages (e.g. in ModelBus or MOFLON) are used to describe data-transfer. A dedicated overview of the tool chain is not available and neither workflows nor model transformations provide an overview of the tool chain for several reasons:

- The workflow language and the model transformation language each capture a partial view of the tool chain. Each view describes the tool chain from the perspective of the respective integration aspect. An overview of the tool chain, however, needs to consider all relevant integration aspects.

- To describe a tool chain, several artifacts representing each integration aspect are needed, for example several instances of model transformations to describe the data-transfer aspect and several workflows to describe the process integration aspect. An overview of the artifacts.

This disintegrated view might be sufficient for the execution of the tool chain, but development of a tool chain could be better supported by an additional overview of different integration aspects and artifacts used for describing a tool chain.

Domain-Specific Description

CORSET provides a domain-specific description for the integration of web applications and focuses on the control, process and platform integration aspects. ModelCVS provides a domain-specific pattern-based model transformation language for the data integration aspect. The surveyed approaches provide domain-specific descriptions for tool chains, but these cover only a limited set of integration aspects.

The descriptions provided by the other surveyed tool integration approaches use general purpose languages for representing specific integration aspects, such as a MOF-conform model transformation language in MOFLON. Also, languages from other domains are borrowed for describing tool integration aspects, for example ModelBus uses BPEL [127] from the business process domain.
CHAPTER 4. STATE OF THE ART

Since the various approaches focus on a subset of the integration aspects, missing integration aspects need to be realized using source code. In MOFLON the data-transfer is described using a transformation language, but control or process integration is not supported. jETI supports control, process and platform integration, but no explicit language for data integration is available. The unsupported integration aspect of the respective approach can be added, but requires the use of source code.

Graphical Representation to Support Communication

The surveyed approaches provide graphical representations for dedicated integration aspects and thus for parts of the tool chain. ModelBus and jETI provide dedicated graphical representations for process integration. MOFLON and ModelCVS provide dedicated graphical representations of the model transformations for data-transfer. However, a graphical representation of an overview of the tool chain including all relevant integration aspects has not been found among the surveyed approaches.

4.3.2 Characterization of the Development of Tool Chains

In this section, the development support for specification, analysis and synthesis provided by the surveyed approaches is characterized according to the criteria introduced in Section 3.2.

Specification Support: Tool Chain

To support the specification of tool chains built from web-applications, CORSET allows recording sequences of user interactions with web-applications. For the other surveyed approaches, the specification of a complete tool chain is assumed to be performed manually.

Specification Support: Tool

The specification of the data and services of a newly integrated tool is a manual, code-centric task for all surveyed approaches. Semi-automated specification support is available when an existing tool adapter is reused. For service-oriented approaches that describe their services using WSDL (Web Service Description Language) [58], such as ModelBus, standard service discovery can be used for obtaining a specification of the services in the interface of the tool adapter. However, only a specification of the tool services and no specification of the tool data can be discovered. The tool data, e.g. in the form of a metamodel, still needs to be manually specified in ModelBus. For OSLC, no automated support for specification is available.
4.3. CHARACTERIZATION OF TOOL INTEGRATION APPROACHES

Specification Support: Other Tool Chain Parts

The specification of tool chain parts such as format conversions and model transformations requires a substantial manual programming effort. In model-based approaches, such as ModelBus and MOFLON, the model transformation rules need to be specified manually. In service-oriented approaches, such as jETI and OSLC, data integration is unsupported.

Analysis Support: Non-Functional Properties

Non-functional properties of tool chains have not received much attention, a finding that is supported by a systematic literature review [164]. The analysis of non-functional properties is not explicitly supported by the surveyed approaches, thus it can be performed manually, or it is unsupported.

Analysis Support: Verification

While generic verification techniques can always be applied on the source code realizing a tool chain, this type of verification support does not take the domain-specific properties of tool chains and the specific requirements of tool chains into account. The specific requirements of tool chains include the development process supported by the tool chain and the specific tools used. Verification of the specific requirements of tool chains is not supported by the surveyed approaches.

Synthesis Support: Tool Chain

As a purely service-oriented approach, jETI provides support for the synthesis of tool chains, which cover control, platform and sequence-based process integration. However, to support embedded systems development, all integration aspects need to be supported by the synthesized tool chain.

If a description of a tool chain was available, which covers all integration aspects, this description could be used for generating source code and infrastructure configurations for a comprehensive tool chain. The surveyed approaches do not provide synthesis support for all integration aspects of a complete tool chain.

Synthesis Support: Tool

MOFLON provides semi-automated code generation for a skeleton of a MOF-compliant repository access method, but it is limited to tool data, tool services are not supported. OSLC does not have a synthesis mechanism, but the Lyo\textsuperscript{1} tool provides support for producing an OSLC-compliant implementation, based on annotations in Java source code. For the other approaches, the tool chain parts need to be created by manual programming, which requires knowledge in the implementation technologies.

\textsuperscript{1}http://www.eclipse.org/lyo
4.3.3 Summary

Out of the reviewed state of the art approaches, several tool integration frameworks were selected for a closer analysis, since the goals regarding description and development support are aligned to the work in this thesis. The selected tool integration approaches were analyzed according to the criteria selected in Chapter 3.

The descriptions provided by current approaches are limited in the concerns covered and in the aspects of integration covered. Most approaches describe several – but not all – aspects of integration. The aspects are described separately, so an overview that expresses the relationships between the aspects is difficult to obtain. Tool chains are described using domain specific languages covering a subset of the integration aspects, general purpose modeling languages, source code or specific languages for other domains, such as languages for business processes, which contributes to the accidental complexity of the description of tool chains. While the focus of model-based tool integration is on the explicit description of tool data and its relation (expressed as models, metamodels and transformations), the entire tool chain is not explicitly modeled. While all approaches have an implicit integration architecture, this architecture is either not made explicit or the architecture is purely conceptual and no explicit link between the architecture and the implementation exists. In addition, the support for the specification, synthesis and analysis of tool chains could be improved by automated methods that take the specific properties of the tool integration domain into account.

The approach presented in this work intends to fill these gaps. After presenting the approach (Sections 5 and 6), we will evaluate it using the same criteria to check if the gaps could be filled (Section 7). To fill the gaps, we envision the following approach. Support for the development of a tool chain could be provided by a holistic description of the architecture of the tool chain. By describing the concerns of tool chains at an architectural level of abstraction, it could be possible to achieve an overview of all integration aspects and provide an information model for the detailed descriptions of different integration concerns. With a graphical representation, such a description of the tool chain could be used as a boundary object for communication among the stakeholders of the tool chain. The description can also serve as a basis for the automated support for specification, synthesis and analysis of tool chains and could have a central, coordinating role for the different activities of tool chain development.
Part II

Overview of Results and Evaluation
Chapter 5

A Modeling Language for Tool Chains

This chapter explains the approach used to answer RQ2 (*How can tool chains be described systematically?*). The approach aims to create a modeling language for tool chains, the Tool Integration Language (TIL). First, the objectives for the language are set (Section 5.1), then the relevant language concepts for the domain are elicited (Section 5.2), and the language is introduced in terms of abstract syntax, concrete graphical syntax and semantics (Section 5.3). A process for systematically developing tool chains based on the language is also proposed (Section 5.4). More details are available in the Appended Papers A, B and C.

### 5.1 Objectives for the Tool Integration Language

The objectives for TIL are set, to overcome the gaps identified by the characterization of state of the art approaches presented in Section 4.3. Thus, the objectives for TIL are expressed using the criteria for the description of tool chains introduced in Section 3.1:

- Coverage of the *architectural integration concern*, with explicit links to semantic, structural and technical concerns.
- Coverage of all relevant *integration aspects*.
- An *overview* of the tool chain.
- An explicit and *domain-specific description* of the tool chain that is independent of implementation technologies.
- A *graphical representation* of the language that is consistent with the implementation of the tool chain, to support communication among the stakeholders of the tool chain.
5.2 Domain Analysis for Tool Chains

We follow a systematic approach for creating a domain-specific modeling language for tool chains. In a first step, a number of domain concepts for tool integration are identified using domain analysis [71]. In a second step, the identified domain concepts are mapped to architecture concepts. Due to the objective of TIL to focus on the architectural concern, domain concepts are mapped to appropriate architectural concepts that are required for any architectural description.

For the first step, a top-down approach is employed and tool integration is broken down into successively smaller building blocks. A starting point is the definition of the scope of tool integration [11, 163], which spans several integration aspects: data integration, control integration, platform integration, process integration and presentation integration. Typical realizations of each integration aspect are further analyzed and domain concepts are extracted (emphasized in italic).

- **Data Integration**: Tool data is the product data (of the embedded system) that a specific tool operates on. Tool data can be tightly integrated by data-transfer or it can be loosely linked by traces. For realizing both data-transfer and data-tracing, the structure of the data of each tool (e.g. in the form of a hierarchy) needs to be described.

- **Control Integration**: Control integration allows for invocation of tool services and notification of users. The signatures of the functions of each tool need to be described.

- **Platform Integration**: Tools may be deployed on a local or remote machine. The data and services need to be uniformly accessible, independent of the deployment of the tools.

- **Process Integration**: To support a development process, all previously identified concepts need to be coordinated and combined into a tool chain that is aligned with the development process. The process can be specified as a sequence or it can be based on events triggered by users or parts of the tool chain.

- **Presentation Integration**: A common user interface is provided for all functionality of the tool chain that can be triggered by a user.

The identified domain concepts and their originating integration aspects are summarized in Table 5.1. A more detailed description of the domain analysis can be found in Appended Paper C.

In the second step, the previously identified domain concepts are mapped to the concepts of an architectural description. Since an objective for the language is coverage of the architectural integration concern, the meaning of architecture in the context of tool chains is elaborated in more detail. In general, an architecture covers the “fundamental concepts or properties” [93] of a system. An architecture
5.2. DOMAIN ANALYSIS FOR TOOL CHAINS

Table 5.1: Identified domain concepts and their originating integration aspects

<table>
<thead>
<tr>
<th>Identified Domain Concept</th>
<th>Data-Transfer</th>
<th>Data-Tracing</th>
<th>Control-Notification</th>
<th>Control-Invocation</th>
<th>Platform-Distribution</th>
<th>Process</th>
<th>Presentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tool Chain</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Tool</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>User</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Data Transfer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Tracing</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Notification</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Invocation</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Sequence</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Events</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Signatures of the Tool Services</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Structure of the Tool Data</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

can also be expressed in terms of the “principal design decisions made about the system” [153]. Typical architectural design decisions relate to structure, behavior, interaction, non-functional properties and implementation [153]. Applied to the context of tool chains, the architectural design decisions relate to:

- **Structure**: the tools and their interfaces.
- **Functional behavior**: how the tool chain reacts to events and the order of the activities in a tool chain.
- **Interaction**: how the tools are connected and the nature of these connections.
- **Non-functional properties**: e.g. the cost for realizing the tool chain, the cost savings achievable by the introduction of the tool chain, or the effort related to certification of tools and tool chains.
- **Implementation**: e.g. the use of certain technical space for integration and integration conventions, such as OSLC [141].

These design decisions are captured by architecture descriptions, which are typically expressed using the following basic concepts [153]: configuration, component, connector and interface. The basic concepts of architecture descriptions have also been identified as the shared concepts among several architecture description languages [119].
To cover the architectural integration concern with the tool integration language, the domain concepts for tool integration identified in the first step are mapped to the basic concepts of an architecture description. This mapping of domain concepts to architectural concepts is summarized in Table 5.2 and the resulting language concepts are described in Section 5.3.

Table 5.2: Mapping of the identified domain concepts to architectural concepts and the resulting TIL concepts

<table>
<thead>
<tr>
<th>Identified Domain Concept</th>
<th>Architectural Concept</th>
<th>Language Concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tool Chain</td>
<td>Configuration</td>
<td>ToolChain</td>
</tr>
<tr>
<td>Tool</td>
<td>Component</td>
<td>ToolAdapter</td>
</tr>
<tr>
<td>User</td>
<td>Component</td>
<td>User</td>
</tr>
<tr>
<td>Data Transfer</td>
<td>Connector</td>
<td>DataChannel</td>
</tr>
<tr>
<td>Tracing</td>
<td>Connector</td>
<td>TraceChannel</td>
</tr>
<tr>
<td>Notification</td>
<td>Connector</td>
<td>ControlChannel</td>
</tr>
<tr>
<td>Invocation</td>
<td>Connector</td>
<td>ControlChannel</td>
</tr>
<tr>
<td>Sequence</td>
<td>Component</td>
<td>Sequencer</td>
</tr>
<tr>
<td>Events</td>
<td>Part of Connector</td>
<td>Events of a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a ControlChannel</td>
</tr>
<tr>
<td>Signatures of the Tool Services</td>
<td>Interface</td>
<td>ToolAdapter Metamodel</td>
</tr>
<tr>
<td>Structure of the Tool Data</td>
<td>Interface</td>
<td>for Services</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ToolAdapter Metamodel</td>
</tr>
<tr>
<td></td>
<td></td>
<td>for Data</td>
</tr>
</tbody>
</table>

5.3 The Tool Integration Language

In this section, we describe the design of a language that is built from the identified domain concepts. The language is designed in such a way that the concepts of the language are composable with each other, i.e. that a tool chain can be described as a composition of instances of the language concepts. The resulting language, the Tool Integration Language (TIL) can be considered both a domain-specific language [123] and an architecture description language [93]. In the following, TIL is briefly introduced in terms of abstract syntax, concrete syntax and semantics. A more detailed description can be found in Appended Paper C and in the TIL Report [29]. The abstract syntax of TIL is presented as a class diagram in Figure 5.1. The graphical concrete syntax of each language concept is introduced by a simple example in Figure 5.2, the concrete mapping function, which maps abstract to concrete syntax, is defined by corresponding circled numbers 0..7 in Figure 5.2 and the following text. This section also briefly introduces the semantics of TIL concepts.

A ToolChain 0 provides a container for instances of TIL concepts. An instance of the ToolChain concept describes the tool chain by the composition of its contained instances of TIL concepts.
5.3. THE TOOL INTEGRATION LANGUAGE

Figure 5.1: A class diagram illustrating the abstract syntax of TIL

Figure 5.2: A simple TIL model illustrating the graphical concrete syntax of the language concepts
A ToolAdapter ① is a software component that describes the role of a tool in the tool chain by exposing the services and data of the tool, which are relevant for the specific role. Exposing the services of a tool enables control integration. Exposing the data of a tool enables data integration. A ToolAdapter makes two kinds of adaptation: (1) It adapts between the technical space of the tool and the technical space of integration for both data and services. (2) It adapts the structure of data and the signature of services available in the development tool to the data structure and service signatures defined by the ToolAdapter metamodel.

Each ToolAdapter has two associated ToolAdapter metamodels: one that specifies the structure of the exposed tool data and another that specifies the signature of the exposed services. In addition to the services defined in the metamodel, all ToolAdapters provide the default services activate to start the tool, injectData to load data (which is an instance of the ToolAdapter data metamodel) into the tool and extractData to access the tool data (as an instance of the ToolAdapter data metamodel). The ToolAdapter metamodels serve as an interface specification for the ToolAdapter and describe which data and services of the tool are exposed. See Appended Papers C and D for more information on the structure of the ToolAdapter metamodels and for examples.

Subtypes of ToolAdapters are defined, such as a GeneratedToolAdapter, whose service implementation is generated based on the specification provided by the ToolAdapter metamodels; a DiscoveredToolAdapter, which is specified by its deployment address, allowing for discovery of the services and binding to the tool chain; and a Repository ⑦, which provides storage and version management, e.g. a ToolAdapter for Subversion [144].

A DataChannel ⑤ describes the possibility to transfer and transform data from a source ToolAdapter to a target ToolAdapter at the run-time of the tool chain; it is a directed connection. The data originates from the source_service of the source ToolAdapter (default service: extractData), is transformed and is finally received by the target_service of the target ToolAdapter (default service: injectData). A model transformation is attached to the DataChannel; the source and target metamodels of the transformation need to match the respective data metamodels of source and target ToolAdapters.

A TraceChannel ⑥ describes the possibility to establish trace links between the data of two ToolAdapters at the run-time of the tool chain; it is an undirected connection. A TraceChannel is a design-time representative for a number of trace links at run-time. At design-time one can specify the type of data that can be linked by traces. The endpoints of the traces can be restricted to a subset of the tool data by specifying the source_service and target_service (default service: extractData), which provide the data. At run-time, these services provide a list of all the source and target elements that are offered as endpoints for specifying a trace.

A ControlChannel ② describes an invocation or notification, it is a directed connection originating from a source component and ending in a target component. If the target of the ControlChannel is a ToolAdapter, the ControlChannel denotes the invocation of a tool service; if the target is a DataChannel, the data-transfer is
executed; if the target is a TraceChannel, a dialog for creating traces is presented. If the target is a User, it denotes notification of the User. A condition for the execution of the ControlChannel can be specified by a guard expression. A service of the source component, called \textit{source\_service} (default value: \textit{activate}), can be specified as the event that triggers the ControlChannel. The invoked service in the target component is specified as the \textit{target\_service} (default value: \textit{activate}) of the ControlChannel.

A \textbf{Sequencer} describes a sequence of invocations or notifications. When a Sequencer is activated by an incoming ControlChannel, it activates the outgoing ControlChannels in the specified order. The order is specified by the events (0..n), which are specified as the \textit{source\_service} in the outgoing ControlChannels from the Sequencer. Only after the service executed by the previous ControlChannel is finished, will the next ControlChannel be activated.

A \textbf{User} represents a real-world tool chain user. The concept is used to describe the possible interactions of the real-world users with the tool chain. Outgoing ControlChannels from the User denote the invocation of tool chain services by the real-world user. Incoming ControlChannels to a User denote a notification sent to the real-world user, e.g. by e-mail.

By default, all TIL concepts describe parts of an automated tool chain, however some parts of the tool chain may not need to be automated and are manually integrated. TIL allows marking ControlChannels, DataChannels and TraceChannels as manually executed, in which case they are depicted by dashed lines.

The semantics of TIL is defined in the text above. The denotational (translational) semantics is defined by the model transformation for code generation described in Section 6.6. In addition, compatible formal semantics of the behavior of TIL can be described by a mapping of TIL concepts to networks of finite state machines (FSMs). Initial work on the semantic mapping function for the formal specification of the behavior of TIL and its implementation as a model transformation for \textit{semantic anchoring} \cite{57} has been performed \cite{31}. The TIL models are mapped to UPPAAL-style FSM models, since these models are formally defined and can be simulated and analyzed using the UPPAAL model checker\cite{111}. The model transformation is defined in two steps. The first step defines for each model element in TIL a template of a corresponding FSM\footnote{In fact, UPPAAL supports networks of timed automata. Timed automata are an extension of finite state machines. The mapping from TIL to UPPAAL, however, does not use any of the advanced properties of timed automata, such as clocks.}. The second step instantiates an FSM template for each element of the TIL model, so that the type of the FSM template matches the type of the TIL model element. In addition, the transitions of the FSMs are synchronized using UPPAAL channels reflecting the connections between the model elements in the TIL model.\footnote{For most language concepts the FSM template is defined by the type (language concept), only for ToolAdapters and Sequencers the FSM template depends on the model element.}
5.4 Proposed Tool Chain Development Process with TIL

The intention of this section is to illustrate and explain the development of a tool chain using TIL. For this purpose, one possible tool chain development process is proposed, which uses platform-based design [103] as a reference model – other development processes are possible.

Platform-based design structures the development process into two processes. The application development process maps functional requirements to a platform representation. The architecture implementation process is concerned with mapping the platform representation to an executable representation. A platform serves as a connection between the two processes. Platform-based design is often depicted by an hourglass shape, where the two processes are described by an upper and lower triangle [103]. The structure of platform-based design is applied to tool chain development with TIL, the result is depicted in Figure 5.3. As a modeling language for explicitly representing the tool chain at an architectural level of abstraction, TIL can be considered as a platform in the sense of platform-based design.

![Figure 5.3: Platform-based design of a tool chain with TIL](image)

The role of the application development process (upper triangle) of platform-based design is to map the requirements of the tool chain to a TIL model. The requirements are partly given by the set of tools to be integrated, their data and services, and partly by the product development process, best practices and company policies for tool usage.
The architecture implementation process (lower triangle) is concerned with mapping TIL to an executable representation. The architecture implementation process resolves the concerns in tool chain construction (c.f. Section 3.1.1) regarding all covered integration aspects (c.f. Section 3.1.2). To realize this mapping, the TIL model is refined with further details to handle the semantic, structural, and technical concerns. Model transformations are used for realizing semantics-preserving data-transfer. Metamodels are used for defining the structure of the data of each tool. The mapping between technical spaces is taken care of by the mapping to the implementation, integration conventions (such as OSLC [141]) and the tool APIs.

In the following, the abstract platform-based development process is concretized by an explicit process model for tool chain development. In Figure 5.4 the process for developing a tool chain with TIL and its connection to the run-time of the tool chain is illustrated using the SPEM [135] notation. The development process for tool chains with TIL is structured into five phases: requirements engineering, conceptual design, detailed design, analysis, and implementation. These phases are presented according to the order in which they are traversed during tool chain development. Platform-based design can be related to the concrete tool chain development process (c.f. with the ordinal number of the phases in Figure 5.4) by mapping the application development process to phases 1–3, and by mapping the architecture implementation process to phases 4–6. The complete tool chain development process has the following phases:

1. The requirements of the tool chain are elicited from the selection of tools and from the dependencies of tasks and tool usages in the product development process.

2. In the conceptual design phase, a conceptual model of the tool chain is described using TIL based on the requirements stipulated by the product development process. The conceptual model conveys the overall architecture of the tool chain, including the existing ToolAdapters, Users and connections between the ToolAdapters.

3. The alignment of the conceptual TIL model with the process model can be verified to highlight any intended or unintended discrepancies between tool chain design and the requirements stipulated by the product development process. Depending on the outcome of the analysis, the conceptual design phase can be iterated in order to create a conceptual model which is better aligned with the requirements.

4. In the detailed design phase, the conceptual TIL model is refined by different types of Channels and ToolAdapter metamodels. ToolAdapter metamodels are attached to each ToolAdapter in the TIL model to describe the data and services of the tool, which are exposed by the ToolAdapter. The connections between ToolAdapters and other components are refined by choosing the type of the connector (ControlChannel, DataChannel or TraceChannel).
A model transformation is attached to each DataChannel in the TIL model; it describes the translation of data from the source tool to the target tool. The model transformation can be specified in different ways, either manually or computed based on the information in an ontology or weaving model (see Section 4.2.1). The conceptual TIL model with attached metamodels and model transformations yields a complete TIL model.

5. Additional analyses are possible based on the detailed design. The syntactic correctness of the model can be checked and non-functional properties, such as the development cost of the tool chain, can be estimated. Depending on the outcome of the analyses, the tool chain design can be corrected before proceeding to the implementation phase.

6. In the implementation phase, the TIL model can serve as a blueprint for implementing the tool chain. The code of the tool chain is compiled and deployed.

7. At run-time of the tool chain, the embedded developers use the deployed tool chain, which integrates several embedded systems tools.

There are different stakeholders of the tool chain, who are in contact with the tool chain at different points in the lifecycle of the tool chain. For this purpose, a distinction is made between the design-time of a tool chain and the run-time.

At design-time of the tool chain, the tool chain development process is executed, which involves a process engineer, tool chain architect and tool chain developers. A process engineer may model the product development process that is supported by the tool chain. The tool chain architect defines the conceptual and detailed design of the tool chain. One or several tool chain developers implement the tool chain as software based on the tool chain design.

At run-time, the tool chain software is executed to realize the data-transfer, traceability, invocation and notification to support the product development process of the embedded system. The embedded systems developers have the role of tool chain users.

The role of the tool chain architect has been explicitly introduced to cover the responsibility of specifying, refining and analyzing the architecture of the tool chain. Since TIL allows the tool chain to be described independently of implementation technology, the role of the tool chain architect can be separated from that of the tool chain developer. As the tool chain users, embedded systems developers are familiar with the requirements for the tool chain, but not with their implementation. Thus, embedded systems developers may be suitable candidates to take on the role of the tool chain architect and leave the implementation to dedicated tool chain developers.

---

2Strictly speaking, the run-time is outside the scope of the development process but has been added here for illustrating the connection between the tool chain and the embedded systems developers.
Figure 5.4: Process for developing a tailored tool chain with TIL, illustrated using SPEM [135] notation
developers. This separation of responsibilities is one attempt to resolve the unclear
responsibility for the creation of tool chains observed in industry (see Section 1.2.2).
Support for the Development of Tool Chains

The previous chapter proposed a language for defining tool chain product models and a process model for developing tool chains with various phases. In this chapter, RQ3 is addressed (To what extent can the development of tool chains be supported by automated methods?). Semi-automated development support for the various phases of tool chain development is provided, enabled by the explicit description of the tool chain in TIL. Figure 6.1 provides an overview of the tool chain development process, which extends the manual tool chain development process presented in Figure 5.4 through the following automated support:

- The TIL Workbench, a development environment for tool chains (Section 6.1, Appended Paper C).
- Construction of a conceptual TIL model from an existing SPEM process model (Section 6.2, Appended Paper E).
- Refinement of a conceptual TIL model by adding metamodels to ToolAdapters (Section 6.3, Appended Paper F).
- Refinement of a conceptual TIL model by creating model transformations for DataChannels (Section 6.4, Appended Paper G).
- Correctness check of the tool chain design (Section 6.5.1, Appended Paper F).
- Structural verification of the tool chain design (Section 6.5.2, Appended Paper H).
- Cost analysis of the tool chain implementation (Section 6.5.3, Appended Paper I).
- Code generation of the tool chain (Section 6.6, Appended Papers D and C).
Figure 6.1: Process for developing a tailored tool chain with semi-automated support based on TIL, illustrated using SPEM [135] notation
6.1 A Development Environment for TIL

The TIL Workbench (see Figure 6.2) is intended to support the tool chain architect in specifying a tool chain with TIL. It consists of the TIL language defined as a modeling language and a graphical modeling environment for TIL in Eclipse\(^1\). The TIL Workbench ensures that the specified models conform to the abstract syntax of the language. Some of the automated methods for development support introduced later in this chapter can be executed from within the TIL Workbench.

![Figure 6.2: The TIL Workbench](image)

6.2 Construction of a Conceptual TIL Model

In early design phase of the tool chain development process, a conceptual TIL model is created, which only describes the components and connections of the tool chain without any additional details. The conceptual model of the tool chain should be aligned to the product development process and the choice of development tools, so the tool chain can support the integration-related tasks in the development process. Development processes can be modeled for different purposes [64], however here we focus on process models that have been created as a means for documentation, and are expressed by the Software and Process Engineering Metamodel (SPEM) [135].

The following research questions (RQ3.1) guided the work: *What is the relationship between a SPEM process model and the tool chain? How can tool chains be tailored, so they support a specific development process and development practice?*

\(^1\)http://www.eclipse.org
If in addition to describing the tool chain as a model (e.g. using TIL), the process is also modeled (e.g. using SPEM), the relationship between the process model and tool chain model can be described. Possible relationships between SPEM and TIL models are identified and expressed by a mapping of a pattern of SPEM metaclasses to a pattern of TIL metaclasses. This mapping is implemented as a model transformation. Using the transformation and an existing process model, an initial conceptual design model of the tool chain is created. The main benefit of an automated mapping between process models and tool chain models is that an alignment between the design of the tool chain and the process can be achieved. This approach is detailed in Appended Paper E.

6.3 Refinement of ToolAdapters in a TIL Model

The conceptual TIL model needs to be refined by adding ToolAdapter metamodels that describe the data and the services exposed by each ToolAdapter and thus serve as interface specifications. If the ToolAdapter is to be newly implemented, the ToolAdapter metamodels need to be manually specified. However, if a ToolAdapter implementation already exists, has been developed by a third party, is deployed and needs to be reused in a new tool chain, it might be possible to gather the information necessary for the ToolAdapter metamodel from the interface of the deployed ToolAdapter. We study this problem in the context of OSLC-compliant ToolAdapters (i.e. the interface of ToolAdapters follows OSLC conventions), since these may be provided by third parties; for example an OSLC-compliant ToolAdapter for the requirements engineering tool DOORS\(^2\) is available as a plugin.

The following research question (RQ3.2) guided the work: How can the metamodels used for interface specifications of ToolAdapters in TIL be automatically discovered and re-engineered from deployed ToolAdapters?

If an existing, already deployed ToolAdapter is to be reused and integrated into a tool chain, such as an OSLC-compliant ToolAdapter provided by a third party, the integration of the ToolAdapter would be possible on implementation level. In this approach, however, we explore the integration at model level, since a complete model of the tool chain enables correctness checks, analysis of the tool chain and complete synthesis of the implementation. The integration on model-level entails representing the interface of the remotely deployed ToolAdapter by ToolAdapter metamodels. This work explores, how the ToolAdapter metamodels of the remotely deployed ToolAdapter can be automatically discovered and integrated into a comprehensive TIL model of the tool chain.

The approach is depicted in Figure 6.3 and detailed in Appended Paper F. Automated discovery and reverse engineering of the ToolAdapter metamodel is used during the design of the tool chain, when an unknown, remotely deployed ToolAdapter is integrated into the tool chain. The discovered and reverse engineered ToolAdapter metamodel is checked for consistency with the TIL model. If

\(^2\)http://www.ibm.com/software/awdtools/doors
6.4. REFINEMENT OF DATACHANNELS IN A TIL MODEL

Figure 6.3: Reverse engineered ToolAdapter metamodel enables correctness check

the models are consistent, the reverse engineered ToolAdapter metamodel is used for automatic generation of a local proxy for the remote ToolAdapter. The approach allows for the efficient reuse of deployed ToolAdapters in a new tool chain, ensures the consistency between the ToolAdapter metamodel and the deployed implementation, and the consistency between the ToolAdapter metamodel and the TIL model, enabled by the representation of all relevant information on the model level.

6.4 Refinement of DataChannels in a TIL Model

The conceptual TIL model needs to be further refined with detailed specifications for each DataChannel. DataChannels denote the transfer of data from a source ToolAdapter to a target ToolAdapter. The tool data is exposed by the ToolAdapter in the form of a model that conforms to the ToolAdapter metamodel. If the metamodels of source and target ToolAdapters are the same, the data can be simply copied between the ToolAdapters. In the more common case that the metamodels are different, the data needs to be transformed before it can be transferred to the target ToolAdapter. For this purpose, TIL offers the possibility to link a model transformation to each DataChannel. The model transformation converts the data between the metamodels of source and target ToolAdapters. The following research question (RQ3.3) guided the work: How and under which circumstances can the detailed specification of DataChannels be computed automatically?

Typically, the details of a DataChannel are manually specified in the form of a model transformation, which requires time and effort. Especially if the requirements for the tool chain are still changing and prototypes of a tool chain are developed, an automated approach for the specification of model transformations can be valuable. In this setting, the intention is to rapidly and automatically create a first prototype of a model transformation, which can be manually refined later on.
CHAPTER 6. SUPPORT FOR THE DEVELOPMENT OF TOOL CHAINS

Under certain conditions it might be possible to provide support for specifying a prototype model transformation automatically. The TIL model contains relevant information for generating the transformation, such as its execution direction and both its source and target metamodels. This information is not sufficient for an algorithmic approach, but a heuristic approach for prototyping model transformations can be realized. With the assumption that similar metaclasses of the metamodels of source and target ToolAdapters should be mapped, a model transformation can be computed using heuristics. As a measure for the similarity of the metaclasses, the similarity of the reference structure and the names of the metaclasses are used.

The heuristic identifies corresponding elements between the source and target metamodels using similarity flooding [120] and the Levensthein distance [112]. The result of the algorithm is a matching table [73], which maps elements of the source metamodel to elements in the target metamodel. The matching table is automatically refined into a matching model by taking the containment hierarchy of the target metamodel into account. The model transformation T1 is a prototype, which may require manual refinement. Based on the corresponding elements that are specified in the matching model, a model transformation T1 is produced, which maps the data between the source and target ToolAdapters. T1 is produced by a higher-order model transformation T2 based on the information in the matching model. While matching algorithms have been applied previously [73], we focus on the applicability and evaluation of a matching algorithm in the context of prototyping tool chains. This step is detailed in Appended Paper G.

6.5 Analysis of TIL Models

The analysis of a tool chain design is intended to support the tool chain architect when designing a tool chain. An advantage of using an explicit model-based description of the tool chain is the possibility for early analysis. Early analysis allows for evaluating different designs of the tool chain and especially allows finding problems during design that would be more expensive to correct if discovered later [94]. Instead of applying generic, existing analysis techniques, we here focus on domain-specific analyses that make use of the additional knowledge of the domain of tool integration, which is encoded in TIL models.

6.5.1 Correctness Checks of the Tool Chain Design

Correctness checks are used to detect specification errors in TIL models. Syntactic correctness of the TIL model is checked by the TIL Workbench when a TIL model is created. In addition, the following checks for semantic correctness are performed. A TIL model provides language concepts for both specifying service signatures and invoking the services. All service calls need to be consistent with their respective specification. Correctness checks compare the usage with the specification. The services and data structures are specified in the ToolAdapter metamodels and are
6.5. ANALYSIS OF TIL MODELS

used in the ControlChannels. The TIL model is checked for correctness by analyzing whether all service usages in the ControlChannels comply with their definitions in the ToolAdapter metamodels. The correctness check in the current implementation checks whether the used services are defined in the ToolAdapter metamodels by using the name of the services. Future work on the implementation could also take the parameters of the services into account.

Since the consistency between the metamodels for ToolAdapters and the TIL model has been checked at the model level, the consistency for the generated implementation can be ensured, as depicted in Figure 6.3 on p. 59.

6.5.2 Early Structural Design Verification of Tool Chain Design

In general, design verification checks if the design fulfills the specified requirements. The requirements for a tool chain are provided by the selection of tools, the product development process and additional information. Here the verification effort focuses on structural design verification, which is concerned with the extent to which the structure of the design of the tool chain is aligned to the structure required by the product development process. Early verification of tool chain design can detect possible misalignments between the structure of the product development process and the structure of the tool chain, when corrections are still relatively simple and cheap. By automating the early verification of tool chain design, it can be performed repeatedly with little effort, supporting the iterative refinement of tool chains. The work was guided by the following research question (RQ3.4): How can the alignment of a tool chain to a specific development process be determined?

The alignment of the design provided by a TIL model is checked against the requirements provided by a SPEM model [136]. This alignment can be expressed by a mapping of a pattern of SPEM metaclasses to a pattern of TIL metaclasses. Even if the conceptual model has been constructed based on a SPEM model as described in Section 6.2, unintended changes might have been introduced by manual refinements. The verification produces a list of misalignments and a measurement indicating the degree of alignment between the tool chain and the product development process using precision/recall metrics [158], where a tool chain that is well-aligned to the process model has both a high degree of precision and a high degree of recall. This approach is described in more detail in Appended Paper H.

Structural design verification is only one part of a comprehensive design verification, since other requirements besides the structure, such as the behavior and non-functional requirements, need to be checked as well. Even a comprehensive design verification is a complement – not a replacement – to testing and verification of the final implementation.

6.5.3 Early Cost Analysis of Tool Chain Design

When a tool chain is constructed, it is important to estimate both the cost of its development and the potential savings achievable when the tool chain is used. These
estimates can be used to compare alternative tool chain designs. The following research question (RQ3.5) guided the work: *How can the development costs for tool chains and the cost savings of tool chains be estimated to allow or the comparison of tool chain designs?*

A cost-efficiency model for tool chains is proposed that quantifies and relates the development costs of tool chains to the cost savings expected from using the tool chain. The net value of a tool chain is given by the cost savings that occur for each product development project that uses the tool chain minus the development cost of the tool chain.

\[
\text{netvalue}_{TC} = \text{costsavings} \times \#\text{uses} - \text{cost}_{TC}
\] (6.1)

Equation 6.1 describes the cost-efficiency model and its main variables, which can be determined by three separate estimates. The variable \(\text{costsavings}\) describes the estimated cost savings for each development project that is executed with support of the tool chain. The achievable savings are dependent on the current tool integration situation and the feature set of the new tool chain. The variable \(\#\text{uses}\) is an estimate of the number of product development projects that are executed by using this tool chain. The variable \(\text{cost}_{TC}\) is the estimated cost of tool chain development, which depends on the feature set of the tool chain. The Constructive Cost Model II (COCOMO II) [46, 47] is used to estimate the \(\text{costsavings}\) in the product development process of the embedded system and a second instance of COCOMO II is used to estimate the \(\text{cost}_{TC}\) of developing the tool chain. By relying on the estimation made by the COCOMO II model, the \(\text{netvalue}_{TC}\) of the tool chain can be estimated.

The calculation of the net value allows a break-even analysis to be made, and can support industrial users when comparing alternative tool chain designs. For such a decision, many aspects need to be considered and cost is only one of them, albeit an important one. The model can help to identify an appropriate feature set for the tool chain and find a balanced tradeoff between the cost savings and the cost of development. This approach is described in more detail in Appended Paper I.

### 6.6 Code Generation from TIL Models

To support the implementation phase of the tool chain development process, the TIL approach provides a code generator. For any correct TIL model the code generator synthesizes a corresponding implementation automatically. TIL is designed to be independent of any particular implementation technology and thus code for different implementation technologies could be generated for a TIL model. For the purpose of showing that code generation is feasible, a particular implementation technology was chosen as a target platform and a code generator was built for it. The work was guided by the following research questions (RQ3.6): *How can the
implementation of a tool chain be synthesized from a tool chain model? To what extent can the tool chain implementation be generated automatically?

The generated implementation is realized using Java and the Service Component Architecture (SCA) [128], a component model for implementing and composing services. The implementation is compliant to the principles of OSLC [141], which stipulates technical conventions for tool integration.

A tool chain described in TIL is realized as an SCA composite, which is a composition of SCA components. Each model element of a TIL model (except those of type ControlChannel) is realized as an SCA component with an implementation in Java. The realization of the SCA component and implementation depends on the properties of the model element and the language concept it is an instance of.

The generator maps a TIL model to an implementation and is defined as a set of mapping rules, as depicted in Figure 6.4. Specific mapping rules for each TIL language concept have been developed as model-to-text transformations, which produce SCA components, Java code and corresponding OSLC descriptions.

![Figure 6.4: Overview of the TIL generator for different language concepts and the generated artifacts. For a detailed description of the technical terms, see Appended Paper C.](image)

The generator creates a part of the implementation of the tool chain and thus reduces the amount of manual work necessary for implementation. Code generation can be completely automated for the language concepts ToolChain, User, Sequencer, Repository, ControlChannel, DataChannel and TraceChannel, as ex-
plained in Appended Paper C. Code generation for ToolAdapters is partially automatized as described in Appended Paper D. In the following, we illustrate code generation for the two language concepts DataChannel and ToolAdapter.

For the DataChannels, it is assumed that a model transformation description in any of the supported model transformation languages (ATL [96], Henshin [44], Acceleo MTL [134] or medini QVT-R [137]) is linked to the TIL model. The code generator for DataChannels creates the code for extracting the source model from the source ToolAdapter, invoking the transformation engine with the supplied model transformation description and the source model, and injecting the target model into the target ToolAdapter.

The implementation of a ToolAdapter consists of an external and an internal part, as illustrated in Figure 6.5. The external part consists of the interface and code for providing the tool data and services which are compliant to OSLC conventions. The internal part of the ToolAdapter realizes the services by communicating with the integrated tool. The generator ensures that the implementation of the external part of the ToolAdapter complies with relevant OSLC conventions; the developer still needs to manually develop the internal part of the ToolAdapter, which is responsible for interacting with the tool via the interface technology of the tool, e.g. the API or file format. For specific interface technologies used by the tools, the communication with the tools can be supported by templates (e.g. for Eclipse, RMI or command line tools). In this way, even the complete implementation of the ToolAdapter can be generated for specific interface technologies, as explained in detail in Appended Paper D.

Figure 6.5: Implementation of a ToolAdapter
Chapter 7

Evaluation of TIL

In this chapter, an evaluation of TIL and its automated development support is presented, which uses the criteria selected in Chapter 3 regarding the description of tool chains (Section 7.1) and the development of tool chains (Section 7.2). The details of the evaluation in the form of qualitatively and quantitatively assessed case studies can be found in the appended papers.

7.1 TIL for the Description of Tool Chains

7.1.1 Coverage of Integration Concerns

The coverage of integration concerns of TIL according to the reference model described in Section 3.1.1 is assessed. As an architecture description language, TIL primarily describes the architectural concern of a tool chain. At the same time, there are dedicated links from the architectural description to more detailed descriptions addressing structural and semantic concerns. The structural concern is covered by ToolAdapter metamodels, which are referenced from the ToolAdapter concepts in a TIL model. The semantic concern is covered by semantics-preserving model transformations, which are referenced from the DataChannel concepts in a TIL model. For technical concerns, there is a need to differentiate between the concerns of integration technology and the technical concerns for accessing tool data. Technical concerns regarding integration technology, such as OSLC-compliance and service-oriented programming, are hidden from the tool chain architect and do not need to be explicitly described, these are handled by the code generator. Technical concerns regarding the access of tool data are described by source code in the internal part of the ToolAdapter.

Even though TIL covers a range of concerns, it mainly focuses on the architectural concern. In the state of the art survey (see Section 4.3), OTIF is the only approach which recognizes the importance of the tool chain architecture. OTIF identifies high-level tool integration patterns, namely *Integrated Datamodel* and *Process Flows* [101]. In the *Integrated Datamodel* pattern, the integrated tools
are connected to a central database, which uses a pivot metamodel as its schema. Bidirectional transformations are used to connect each tool to the schema of the database. In the Process Flow pattern, tools are connected to a bus, called Model-Backplane, using transformations. The transformations are executed according to a process description. In the following we evaluate, to which extent TIL can be used to model both patterns.

Figure 7.1 expresses the pattern of the Integrated Datamodel [101] in TIL. A ToolAdapter is used to represent the Integrated Datamodel and the schema of the Integrated Datamodel is assigned as the ToolAdapter metamodel. If an additional tool is added to the tool chain, two DataChannels with attached model transformations are required for both import and export of data from the ToolAdapter of the integrated tool to the ToolAdapter of the Integrated Datamodel. Interestingly, the Integrated Datamodel pattern does not describe any control integration. Using TIL, an adequate control integration could be added to the pattern.

![Figure 7.1: Modeling the Integrated Datamodel pattern [101] in TIL](image)

Figure 7.2 models the pattern for Process Flows [101] in TIL. A uni-directional DataChannel is used to represent the data-transfer in the Model Backplane of OTIF. In addition, a User and ControlChannels are used to represent control-integration.

![Figure 7.2: Modeling the Process Flow pattern [101] in TIL](image)

The architecture description language TIL is expressive enough to instantiate both patterns. With the language, it is furthermore possible to instantiate variants of the patterns, mix both design patterns or instantiate completely new patterns. TIL provides more flexibility at design-time and allows abstracting further from
7.1. TIL FOR THE DESCRIPTION OF TOOL CHAINS

implementation details than in the original description of the patterns [101]. For example, TIL simplifies the description of event registrations.

Since TIL provides an architectural description, TIL can be compared to other architecture description languages. Can existing ADLs (see Section 4.1) be used to describe tool chains? Existing ADLs are compared and classified in several surveys [60, 72, 119], but some of the surveyed ADLs are not in use any more. Some ADLs are only used for communication, and additional support, such as code generation and analysis, is not available. Medvidovic [118] explains the short lifespan of some ADLs by their focus on technological aspects and their lack of consideration of the application domain, business context and development context. As a result, Medvidovic proposes domain-specific ADLs [118] focusing on a particular domain and business context. This finding is confirmed by another report on the use of ADLs in practice, which emphasizes that ADLs need to be aligned more closely with industrial software practice [167].

TIL addresses the identified shortcomings of generic ADLs by proposing a domain-specific architectural modeling language for tool chains. TIL considers the business context by addressing cost estimation for tool chain construction. TIL addresses the industrial software practice and development context by producing OSLC-compliant implementations. The application domain is considered by using domain analysis for selecting language concepts from the domain of tool integration. Thus the domain-specificity of TIL provides additional value compared to general purpose ADLs.

7.1.2 Coverage of Integration Aspects

By design, TIL covers a range of tool integration aspects. TIL represents each integration aspect with dedicated language concepts. Each language concept provides more detailed information in the form of properties. Some language concepts contain references to more detailed descriptions, which are expressed in other languages, such as model transformations and metamodels.

- The data-transfer aspect is explicitly modeled by DataChannels, which are independent of the approach used for data conversion. The data conversion of the DataChannels could be realized by pivot metamodels or ontologies, but for practical reasons, we limit the realization of data conversions to model transformations. At the tool chain design-time, the model transformations are specified and linked to the TIL model, at tool chain run-time, the model transformations are executed.

- The data-tracing aspect is explicitly modeled by TraceChannels. At design-time the possibility to create traces can be specified and limited to certain types of data elements exposed by the connected ToolAdapters. The trace instances can be created manually by the tool chain user at run-time of the tool chain. In addition, traces can be created automatically as by-products of model transformations.
• The \textit{control-invocation aspect} is expressed by ControlChannels targeting a ToolAdapter, DataChannel or TraceChannel.

• The \textit{control-notification aspect} is expressed by ControlChannels targeting a User.

• The \textit{process integration aspect} is expressed by the ToolChain language concept, which is a composition of all other TIL concepts. The relationship between the process and the tool chain can be described both in an event-triggered manner using ControlChannels and in a sequence-based manner using Sequencers.

• The \textit{platform-distribution aspect} is partly abstracted by the code generator, and partly expressed by the language concept DiscoveredToolAdapter. Remotely deployed ToolAdapters can be integrated into a new TIL model using the DiscoveredToolAdapter language concept and the discovery functionality described in Section 6.3. By mapping the TIL model to an implementation based on SCA (see Section 6.6), the code generator realizes the tool chain described by the TIL model as a distributed system.

• The \textit{presentation integration aspect} is expressed by all the ControlChannels that originate from Users, since these represent the interactions of the real-world users with the tool chain. Code generation produces a website as a common user interface, which allows the real-world user to interact with the tool chain, as specified by the User concept in TIL. Additional forms of presentation integration are part of future work (see Section 9.2.6).

\subsection{Overview of the Tool Chain}

TIL can provide an overview of the tool chain by representing tool chain at the architectural level of abstraction. At this level of abstraction, each tool integration aspect is represented by specific language concepts of TIL (see Table 5.2). In addition, two mechanisms are used:

• \textit{Composition:} To describe a tool chain in a TIL model, instances of the TIL language concepts are composed. The composition provides an overview of the tool chain and its various integration aspects.

• \textit{Reference:} Detailed descriptions for each of the integration aspects are referenced from the TIL model. References from a TIL model to detailed descriptions of structural concerns (i.e. metamodels) and semantic concerns (i.e. model transformations) provide an information model for the tool chain and provide an overview of the various detailed descriptions. By following the references, it is possible to “zoom in” on specific parts of the tool chain to obtain additional details.
From the overview provided by a TIL model, specific views of the tool chain can be generated. An example is the implementation view, which is created by the code generator.

### 7.1.4 Domain-Specific Description

The language concepts of TIL are the outcome of a domain analysis for tool chains (see Section 5.2). TIL is thus a domain-specific description. A TIL model is independent of the implementation technology used in the tool chain; the source code is introduced by the code generator. TIL depends on model transformations as detailed descriptions of the DataChannel. However, no particular transformation engine is assumed, since several alternative model transformation engines are supported (see Section 6.6). Only the internal part of the ToolAdapters needs to be described using implementation technology.

### 7.1.5 Graphical Representation to Support Communication

TIL has a graphical concrete syntax and is supported by a graphical modeling environment. This graphical representation of the tool chain can be used to communicate with the various stakeholders of a tool chain (see Section 1.2.2).

In Appended Paper C, a case study is used to determine if TIL is sufficiently expressive to describe a tool chain, which was provided by an industrial partner in the form of text and sketches. The need to express the features of this tool chain using the given structure of the modeling language forced us to be more precise in the description of the tool chain, for example regarding the directionality of the transformations or the desired granularity of the tool data. In contrast to a description of the tool chain in source code, the architecture description of the tool chain with its graphical representation can be used as a boundary object. Different stakeholders, such as the tool chain architect and tool chain users, can relate to the boundary object, which thus provides a common ground for communication. Other studies confirm that architecture description languages can be used as boundary objects [143].

### 7.2 TIL for the Development of Tool Chains

Due to its focus on tool chains, the development support provided by TIL can take the specific properties of tool chains into account. The development support for the various phases of tool chain development has been evaluated separately, as described in the appended papers, and briefly summarized here.

#### 7.2.1 Specification Support: Tool Chain

A TIL model of a tool chain can be specified manually or with automated support. For the manual specification of the tool chain, the TIL Workbench provides a
Automated support for specifying a conceptual TIL model is provided, based on a model of the development process described in SPEM. An existing SPEM model is used as a source for the structure of the tool chain and parts of the tool chain design can be deduced from the development process.

The automated approach was applied in two case studies. The SPEM model of the first case-study was created according to the development process of an industrial partner. The SPEM model of the second case-study was taken from the official methodology description of AUTOSAR R3.2 (Automotive Open Software Architecture) [15]. In both case studies, a cohesive design of a tool chain could be created based on existing SPEM models, and the resulting tool chain model was actually aligned to the process. A detailed description of the evaluation is available in Appended Paper E.

### 7.2.2 Specification Support: Tool

The tool chain architect is supported in specifying the ToolAdapters manually and with automated support. For the manual specification of the ToolAdapter, the TIL Workbench is provided, which offers support for specifying the ToolAdapter metamodels.

The TIL approach supports the tool chain architect in building the tool chain by facilities to include remotely deployed ToolAdapters into the tool chain. Remotely deployed ToolAdapters are discovered and the ToolAdapter metamodel is automatically reverse-engineered. This allows for a comprehensive model-based description of the tool chain that can be analyzed and for which proxies for the invocation can be synthesized. A case study is designed to discover an existing, remotely deployed OSLC-compliant ToolAdapter for MATLAB/Simulink. This involves reengineering the ToolAdapter metamodel of the discovered ToolAdapter, which allows for integration into a TIL model. The original metamodel of the MATLAB/Simulink ToolAdapter, which was used for specification and code generation, is compared to the discovered metamodel. The case study is described in Appended Paper F.

### 7.2.3 Specification Support: Other Tool Chain Parts

The TIL approach provides automated support for specifying data-transfer in tool chains. Automated support for data-transfer is typically limited to an automated execution of a manually specified data-transfer method. The TIL approach supports the tool chain architect in semi-automatically specifying model transformations as a part of the DataChannels. The algorithm for creating model transformations can be used when the metamodels of the ToolAdapter connected by the DataChannel are similar.

In the general case, the creation of a semantics-preserving model transformation between two given metamodels is difficult, since an understanding of the semantics of the involved metamodels is required. However, the semantics of metamodels is
usually not explicitly specified and thus not available for computation. A heuristic approach can compute a model transformation automatically, only if additional assumptions about the metamodels and their semantics are made. The presented heuristic makes the assumption that structurally similar elements and elements with similar names are also semantically related. This assumption is reasonable when tools with similar functionality need to be connected, e.g. for migration to a new version of the same tool, or for migration to a corresponding tool from another vendor. Another scenario is covered by this assumption as well: the ToolAdapter metamodel represents a view of the data in the tool and can be chosen by the tool chain architect, depending on the viewpoint taken (see Section 5.3). Thus the metamodels for several tools could be created from a common viewpoint, resulting in a similarity of the respective ToolAdapter metamodels.

As a preparation for the evaluation, three different ToolAdapter metamodels were selected, which were partially similar. To evaluate the algorithm, these three metamodels were used to calculate model transformations for all six pairwise combinations. The generated mappings were compared to the respective manually created reference mapping using the precision/recall metric [158]. The generated transformation code for the DataChannels provides a precise mapping (93% precision on average), but does not cover all mappings required by the reference mapping (56% recall on average). Due to the conservative characteristic of the approach (high precision, average recall), the generated mapping can be manually extended into a comprehensive mapping. The generated artifacts can serve as a starting point for manual extensions and refinements of the generated tool chain implementation. A detailed description of the evaluation is available in Appended Paper G.

7.2.4 Analysis Support: Verification

The TIL approach provides automated support for checking the alignment between tool chain design and a model of the development process. The development process that is intended to be supported by the tool chain constitutes a part of the requirements of the tool chain. The alignment check can thus be used as a verification method for the tool chain design.

The automated alignment check between a SPEM model and a TIL model is evaluated using a case study. The alignment check could detect deviations between a process and the supporting tool chain. A deviation should be regarded as a warning flag for tool chain design or process modeling. Expert judgment is needed to determine whether the cause of the inconsistency is in the process model or the TIL model. A detailed description of the evaluation is available in Appended Paper H.

7.2.5 Analysis Support: Non-Functional Properties

Several non-functional properties of tool chains are relevant for analysis, such as performance, safety and cost. We focus here on a method for estimating the cost
of implementing the tool chain and an estimate of the cost savings that are made possible by the use of the tool chain. The resulting cost estimation can be used as a relative measure, to make tradeoff decisions and compare tool chains resulting from different design decisions, and not as an absolute measure.

Work on non-functional properties of tool chains, especially on the economic implications of tool chains, is relevant for industrial practice but has not received much attention from academia [164]. Thus the work on cost estimation for tool chains focuses on the feasibility of automated support based on the TIL description, as described in Appended Paper I. An in-depth evaluation of the cost analysis and an exploration of analysis methods for additional non-functional properties is a part of future work.

7.2.6 Synthesis Support: Tool and Tool Chain

Synthesis support allows for the automated code generation of the tool chain implementation from a TIL model. The effectiveness of the code generator is determined by comparing the effort necessary for developing the tool chain with and without the TIL code generator using the case study described in Appended Paper C: (1) In previous work, parts of the studied tool chain have been implemented manually [8, 32, 39] and are extended to reflect the same case study; the result is referred to as the CM (completely manual) approach. (2) The implementation for the tool chain is generated from a TIL model and is referred to as the TIL implementation. Both qualitative and quantitative evaluations of the case study are used to compare the CM and TIL approaches. The qualitative evaluation shows that writing the manually added code of the TIL approach only requires making a subset of the design decisions necessary in the CM approach. The quantitative evaluation shows that the manually added code in TIL only makes up 20% of the code of a comparable, completely manually written tool chain with the CM approach. The qualitative and quantitative evaluations together provide evidence that the TIL approach can effectively support the implementation of the tool chain.

Besides the representative case study described in Appended Paper C, we also studied extreme values of a critical part of the tool chain, the size of the ToolAdapter metamodel, as described in Appended Paper D. While a typical metamodel has roughly 10 metaclasses, the behavior of the generator was explored for a metamodel that has over 20 times the typical size (complete metamodel of UML\textsuperscript{1}). From this extreme case, some insight for the usage of TIL was gained. While the technology of the automated code generation was scalable and even allowed for complete code generation of the ToolAdapter, the understandability of the large metamodels decreased. The native metamodels of tools, such as the UML metamodel in this case, are large and contain a wealth of information that is typically not relevant in a tool integration scenario. Often, smaller amounts of data are relevant for other tools, so the ToolAdapter metamodel can provide a simplified

\textsuperscript{1}http://www.eclipse.org/uml2
view of the tool’s native metamodel. In addition, a tradeoff was identified between reusing an existing but large metamodel, and creating a new simplified metamodel: if the native metamodel is used, the ToolAdapter does not require any mapping and can be generated completely automatically. However, for the solution with the simplified metamodel, the internal part of the ToolAdapter needs to be developed manually to ensure the mapping between the native and the new metamodels.
Part III

Reflection
Chapter 8

Discussion

In this chapter, we discuss the implications of the TIL approach from several alternative perspectives. We discuss the challenges of tool integration and realize that not only development tools need to be integrated into a tool chain, but also the integration technologies used for realizing tool chains (Section 8.1). We also discuss the properties of tool chains, for which a model-driven approach for tool chain development has advantages (Section 8.2). We explain that tools can have different roles in a tool chain and the implications for the development of ToolAdapters (Section 8.3). Finally, we discuss possible implications of this work for the industrial practice (Section 8.4).

8.1 Primary and Derivative Integration Challenge

Tool integration is motivated by the multitude of unconnected development tools and the need to use data and services across tool boundaries. We call this challenge the primary integration challenge. For solving the different aspects of the primary integration challenge, various integration technologies (tracing tools, transformation tools ...) are typically used. These technologies typically focus on one integration aspect and describe the integration limited to this aspect. A tool chain, however, typically needs to take care of several integration aspects and thus needs to use several integration technologies. The integration of integration technologies is called the derivative integration challenge.

The primary and derivative integration challenges are illustrated in Figure 8.1. While the primary challenge of tool integration is defined as the activity of producing an environment of tools that supports the embedded systems development process, the derivative challenge of tool integration is the activity of producing an environment of integration technologies that supports the tool chain development process.

The derivative integration challenge becomes relevant for larger tool chains, which require integration according to several integration aspects. For these inte-
CHAPTER 8. DISCUSSION

Figure 8.1: Illustrating the Primary and Derivative Integration Challenge

giration scenarios, an overview of the tool chain and its functionality is required. This overview of the tool chain is not sufficiently provided by the surveyed state of the art approaches for tool integration (see Section 4). Current integration technologies focus on solving the primary challenge, while the derivative challenge has not received much attention.

To deal with the derivative integration challenge, both mechanisms to describe the derivative integration challenge for a particular tool chain and mechanisms to build a tool chain from separate integration technologies are needed, which each focus on a particular integration aspect or possibly on a subset of integration aspects. The goal for describing a tool chain is an overview of a tool chain realized with different integration technologies. The goal for building a tool chain is a coordination of the different integration technologies.

8.1.1 Description of Tool Chains

As a first step towards describing the derivative integration challenge, Wasserman [163] identifies five aspects of tool integration, namely data integration, control integration, process integration, platform integration and presentation integration. Each integration aspect is supported by different integration technologies (see Figure 8.1). All the different integration aspects are equally valid and contribute to the comprehensive description of a tool chain. The practice of viewing the tool integra-
tion problem from different angles, resulting in separate views, has its advantages, but is also a challenge. The advantage is that dedicated languages and mechanisms for single integration aspects can be developed independently, each realizing a part of the overall tool chain; the challenge is that it leads to a fragmented description of tool chains. The fragmented description of tool chains in itself would not be a problem, but existing approaches neither provide an overview of the integration aspects nor an adequate way to compose a tool chain from the separate descriptions.

8.1.2 Building Tool Chains

When building tool chains, the challenge is the integration of the fragmented and specialized integration technologies. An example is a tool chain for which the data integration aspect is described by an ATL model transformation and the process integration aspect is described in BPEL. Even though both are domain-specific descriptions for the respective integration aspect, they cannot be readily combined without using source code\(^1\).

8.1.3 TIL Approach to the Derivative Integration Challenge

The derivative integration challenge surfaces both when describing and when building tool chains. The TIL approach offers a possible solution to the derivative integration challenge. Since TIL describes tool chains at an architectural level of abstraction, the different integration aspects can be represented as language concepts. By including different language concepts into a TIL model, several integration aspects of a tool chain can be described and thus an overview of the integration aspects is provided. To illustrate this point, we use the example from the previous section: When a data-transfer is triggered by an event in a process, this can be expressed in the same TIL model by a ControlChannel ending in a DataChannel, as illustrated in Figure 7.2 on p. 66.

The abstract syntax of TIL can be regarded as a pivot metamodel (see Section 4.2.1) for the domain of tool integration. The concepts of TIL serve as a reference for relating alternative integration technologies that cover the same integration aspect and also to relate integration technologies of different aspects among each other.

To build a tool chain that covers several integration aspects, TIL provides a code generator that synthesizes the integration solution from a TIL model. The synthesized integration solution might involve several separate integration technologies. There can be one integration technology for each of the various aspects, or even several integration technologies for the same aspect. For example, several transformation engines are supported for realizing the data-transfer aspect. New integration technologies can be added to TIL by adding a specific generator.

---

\(^1\)A possible solution for realizing the composition is providing a wrapper for the ATL engine with an interface supported by the BPEL engine.
8.1.4 Summary

Paradoxically, the state-of-the-art solutions for tool integration face an integration challenge: The solutions for the challenge of the primary tool integration among development tools for embedded systems, yield another integration challenge, the derivative integration challenge among the integration technologies.

To improve tool integration, not only improved integration technologies for each integration aspect are needed. This allows only for local improvements. To improve tool integration as a whole and to build more powerful tool chains, also an improved integration of the various integration technologies is needed. TIL provides an approach to tackle the derivative integration challenge. By describing tool chains at a higher level of abstraction than each of the integration aspects does, an integrated description for several integration aspects becomes possible. The synthesis mechanism of TIL enables the construction of tool chains, which realize several integration aspects.

8.2 From Model-Based to Model-Driven Tool Integration

Model-based approaches explicitly represent the main artifact that is developed by a model. Accordingly, *model-based tool integration* approaches explicitly model the data of the integrated tools to realize data-transfer among tools (see Section 4.2.1). In contrast, the main artifact in the TIL approach is the tool chain; the tool chain is thus modeled explicitly, in addition to modeling the data of the integrated tools. Furthermore, the explicit model of the tool chain drives the development of a tool chain. Since an explicit model of the tool chain exists, several tasks in the tool chain development process can be supported. Thus TIL realizes what we call *model-driven tool integration*.

Model-driven tool integration has advantages, but also requires an upfront investment in the infrastructure, i.e., the definition of metamodels, transformations and development environments. In the following sections, the circumstances are characterized that justify a model-driven development approach for tool chains and the key features that the model-driven development approach for tool chains needs to expose.

8.2.1 Size of the Tool Chain

Tool chains may need to support many of the tools used for the embedded systems development process. In addition, several integration aspects may need to be taken into account, requiring the use of different integration technologies. These two factors contribute to the size of the tool chain.

---

2Note, that our use of the term requires that the tool chain development process is model-driven. Others use the term for tool integration that supports a model-driven product development process e.g. of an embedded system.
The larger the size of the tool chain, the more leverage can be provided by an explicit model of the tool chain for improved management of the information about the tool chain. A structured description of the tool chain at an appropriate level of abstraction is a means to manage this information efficiently and reduce the accidental complexity.

The larger the size of the tool chain, the more leverage can be provided by automated development support for the tool chain. Especially for larger tool chains, synthesis is preferably automated. Early analysis of the tool chain can help to ensure that the right tool chain is developed, before effort is wasted on implementing a tool chain that cannot fulfill the requirements.

### 8.2.2 Extensive Use of Integration Conventions

The goal of many tool integration initiatives is to create standards and conventions for data and control integration on a technological level (see Section 3.1.1). An example is the OSLC initiative, which proposes the use of RDF for representing tool data and the use of RESTful services for representing tool services. This complicates the development of the components, since they need to be realized using predefined technology, libraries, protocols and formats. This contributes to an increase in development effort, necessary training in the technologies and potential bugs in the implementation of the tool chain.

With a model-driven tool integration approach, implementation technology and conventions are hidden from the tool chain architect. The explicit tool chain model abstracts from implementation details and provides a technology-independent representation of the tool chain reducing the perceived complexity of the problem. The generator takes care of the synthesis of an implementation of the tool chain that is compliant with industrially relevant conventions for integration.

The generator only needs to be written once (for each supported implementation technology) and can then be reused for building many tool chains. The more tool chains are built with the generator, the sooner the additional effort for generator development is amortized.

### 8.2.3 Tailoring

Seamless tool chains with a process scope are ideally built in such a way that they accommodate the development process and the set of tools used. Each company uses a different selection of COTS development tools and even some company-specific tools that are developed in-house. Each company uses a development process which is customized to the needs and circumstances. One-size-fits-all tool chains do not accommodate the requirements of tool chains with a process scope. If tool chains are built from predefined components, some tools used by the company might not be available, such as company-specific tools, leading to tool chains that do not support the complete development process. To support tool chains with a process scope, a model-driven tool integration approach offers the possibility to build tool
chains that are tailored to specific needs, by describing tool chains at a high level of abstraction using a graphical modeling language. The generator supports the efficient construction of a tool chain implementation from the description of the tool chain.

Many European projects (iFEST [2], CESAR [1], MBAT [3]) develop OSLC-compliant libraries of tool adapters, however, there is less emphasis on support for tailoring and composing these components into a customized tool chain. The TIL approach supports specifying the whole tool chain and its components according to custom requirements and is thus complementary to the component-based approach. A combination of both approaches is possible, using the discovery approach described in Section 6.3.

8.2.4 Changing Requirements

A tool chain may evolve over time, as new tools are included into the tool chain, which may require changing existing parts of the tool chain. A particular situation with changing requirements and the subsequent need for flexibility of the tool chain, is the prototyping of tool chains. Prototyping is characterized by frequent cycles of specification, synthesis and testing. The model-driven approach for tool integration makes the most important design decisions concerning the tool chains and its parts explicit and expresses them in the form of a TIL model and the ToolAdapter metamodel. Using an explicit representation of the tool chain, the specification can be tailored to fit the requirements of the tool chain and can simplify the process of changing and evolving the tool chain.

An important factor for dealing with changing requirements is the support for the synthesis of the implementation. The synthesis mechanism assures the consistency between the design and the implementation. Ideally, the synthesis should support iterative development, i.e. manually added code for the internal parts of the ToolAdapter is preserved, when the implementation is regenerated after the tool chain model is updated. TIL supports this feature by realizing the code generation with the concept of protected regions for the user-defined source code that will not be overwritten.

Another important factor for dealing with changing requirements, is support for checking that the tool chain implementation fulfills the requirements. TIL can be used to check the consistency between the structural requirements and the design. If the tool chain changes during prototyping, the changed tool chain needs to be communicated to its stakeholders. A model of the tool chain – as provided by TIL – not only documents and communicates the tool chain graphically, but the model also describes how the tool chain is used.

8.2.5 Summary

Model-driven tool integration can provide advantageous support especially for the case, when tool chains are larger in size, when the use of integration conven-
8.3. ROLES OF TOOLS IN THE TOOL CHAIN

8.3 Roles of Tools in the Tool Chain

A development tool can be used in different ways, and depending on its usage, the tool can take on different roles in the development process or tool chain. A ToolAdapter captures the role of the tool and its ToolAdapter metamodel describes only the data and services, which is relevant for this role. Thus, there can be several ToolAdapters for each tool. TIL supports the creation of these role-dependent ToolAdapters by synthesis methods.

The ToolAdapter model represents the tool data according to the ToolAdapter metamodel. The mapping from the tool data to the ToolAdapter model is not necessarily a projection, but may include an interpretation of the tool data. This allows us to build more specific ToolAdapters. Let us consider the example of a word processing tool, which may in industrial practice take the role of a requirements engineering tool. Without the role-specific interpretation, a ToolAdapter for a word processing tool can only consider the formatting of the document. With the interpretation, a ToolAdapter for a word processing tool may interpret certain text passages of the document as requirements.

Not only can tools take on various roles in a tool chain, but also several tools can take on the same role, which makes them replaceable with respect to that role. Since the ToolAdapter metamodel only depends on the role – not on the specific tool – the impact of tool replacement can be contained, and the rest of the tool chain remains unaffected by the replacement. As a consequence of using role-specific ToolAdapters, tool replacement in a tool chain can be simplified. Support for the replacement of tools is an issue in industry, since the ability to replace a tool ensures the independence of a specific tool and tool vendor.

8.4 Potential Implications for Industrial Practice

The TIL approach offers new opportunities for the industrial practice of embedded systems development. TIL enables embedded systems developers to create tailored tool chains for their different projects instead of relying on one-size-fits-all tool chains. Since tool chains can be tailored to the development process using TIL, the tool chain can better support the embedded systems developers, in their tasks along the development process.

Since the TIL approach allows building tool chains with reduced effort, i.e., relatively fast and at low cost, companies may be in the position to build several tool chains. Each tool chain can take the specific needs of each software development project into account, when creating a new tool chain. Since much of the integration
technology is hidden by the generator and thus the necessary knowledge of integration technologies is reduced, embedded systems developers may be able to build tool chains themselves, instead of buying tool chains from external consultants. At the same time, the tool chain implementation still follows integration conventions and can be reused.

Embedded systems developers themselves best know the requirements for a tool chain that can support their work. If embedded systems developers are closely involved in the tool chain development process, and are not only tool chain users, but also become tool chain architects, this is likely to result in an improved alignment of the tool chain to the industrial practice. Due to this alignment, tailored tool chains may have the potential to increase developer productivity and speed up the development process of the product.

Another concern for industry is the dependence on development tools form a particular tool vendor, even though a better competing tool from another vendor is available. This phenomenon is common among competing technologies and is known as lock-in [9]. TIL reduces the lock-in by role-specific ToolAdapters, which simplify tool replacement with competing tools, as described in Section 8.3.
Chapter 9

Conclusion

In this chapter, we describe the contribution of the thesis by reviewing the research questions (Section 9.1) and outline possibilities for future work (Section 9.2).

This work contributes to the area of tool integration. Tool chains for embedded systems, and specifically their development have been studied. Tool chains are needed for handling the heterogeneity of industrial development tools for embedded systems. In practice, the development of tool chains, which are tailored to the needs of a specific company, is still a code-centric and time-consuming effort. State-of-the-art model-based tool integration approaches create models of the data of the integrated tools, but not an architectural model of the tool chain that captures all relevant integration aspects.

In this work, possibilities for modeling a tool chain as a whole are studied and the resulting implications for tool chain development are evaluated. To model a tool chain holistically and systematically, a domain-specific modeling language for tool chains is created. It describes a tool chain at an architectural level of abstraction, provides concepts from the domain of tool integration, covers all relevant integration aspects, and is independent of particular implementation technology. A graphical representation of the tool chain model can be used for documentation and communication. The modeling language is the cornerstone for semi-automated development support for the specification, refinement, analysis, and in particular the synthesis of a tool chain. According to the model-driven development terminology, we call this approach model-driven tool integration, since an explicit model of the tool chain drives the development.

9.1 Review of Research Questions

The work has been guided by the overall research questions RQ1-3 presented in Section 2.2. Based on the results presented in Chapters 5 and 6 and the evaluation in Chapter 7, the research questions can be answered as follows.
CHAPTER 9. CONCLUSION

RQ1: How can the development of tool chains be systematized and automated?

Instead of developing the tool chain in an ad-hoc manner, the tool chain development can be systematized by regarding the tool chain as a product that is developed according to a model-driven development process. This allows us to systematize the development by both product models describing the developed tool chains and a process model describing the tool chain development process.

To provide the means for a systematic description of tool chains as product models, a modeling language for tool chains is proposed. To describe the tool chain and its parts explicitly, the modeling language is domain-specific and its language concepts are from the domain of tool integration. To describe the tool chain holistically, with consideration of all integration aspects and to reduce the accidental complexity of the technical tool integration concerns, the language is at the architectural level of abstraction.

The modeling language also serves as an enabler for a structured tool chain development process and semi-automated development support for various tasks in the process. The development process can be structured into distinct phases: specifying a conceptual design from the requirements; refining the conceptual design into a detailed design; analyzing the design and code generation from the design. For each phase, specific tasks need to be performed with the product model and semi-automated support for these tasks is provided.

The support for describing and developing tool chains is especially relevant for those tool chains which are larger, are tailored to specific tools or processes, need to accommodate changing requirements, or comply to implementation conventions. Tool chains for embedded systems development typically exhibit all or several of these properties, so the provided support is applicable in this context, as demonstrated in several case studies.

RQ2: How can tool chains be described systematically?

Tool chains can be described systematically by a modeling language that is domain-specific and at an architectural level of abstraction. The language is domain-specific to reduce the gap between the description and the terms used in the domain of tool integration and to cover all relevant integration aspects and concerns. The architectural level of abstraction is appropriate, since it allows for an overview of the tool chain by expressing the most important design decisions for building tool chains. Describing tool chains as domain-specific models on the architectural level of abstraction reduces the accidental complexity and allows focusing on the specification and analysis of the overall tool chain design.

Potential language concepts for the domain-specific modeling language are identified by an analysis of the tool integration domain. The broad scope of the tool integration domain is broken down into several integration aspects. Each integration aspect contributes with domain concepts for describing tool chains. Since the
language concepts originate from different integration aspects, a holistic description of the tool chain can be achieved. By mapping the domain concepts to the basic concepts of architecture descriptions, the languages can be seen as an architecture description language for tool chains. The architecture of a tool chain can be systematically described with the language by composing instances of the language concepts.

The domain-specific architectural modeling language TIL is built according to these principles. TIL is applied in several case studies for systematically describing tool chains.

**RQ3: To what extent can the development of tool chains be supported by automated methods?**

The domain-specific architectural modeling language provides the means for describing tool chains explicitly and systematically as models. A model-driven approach for the construction of tool chains can make use of the tool chain model during conceptual design, detailed design, analysis and implementation. For each of these phases, possibilities for automated development support are proposed and evaluated.

In the conceptual design phase, the overall architecture of the tool chain is determined and described in a conceptual tool chain model. A model of the embedded systems development process that needs to be supported by the tool chain may already be available and describe some of the requirements of the tool chain. Automated support for specifying a conceptual tool chain model can reuse relevant, existing information from models of the development process.

In the detailed design phase, the conceptual tool chain model is refined by detailed specifications of the data and services exposed by each tool and rules for data conversions. When reusing existing, remotely deployed tool chain parts to build a new tool chain, the specifications of the exposed data and services can be automatically re-engineered allowing for consistency checking and early analysis of the tool chain. The creation of prototypical data conversion rules can be semi-automated by a heuristic if the connected tools are similar, which is relevant for scenarios such as data migration or tool replacement.

Early automated analysis can help to ensure that the right kind of tool chain is built and thus prevent wasting effort on tool chains that do not fulfill the basic functional and non-functional requirements. A tool chain model can be automatically analyzed for various properties. The alignment of a tool chain model with a process model can be checked automatically; the correctness of a tool chain model can be checked; and to compare alternative tool chains, the development cost of the tool chain can be estimated and put in relation to the expected cost savings.

Code generation can transform a holistic tool chain model that covers several integration aspects into an implementation, which is compliant with integration conventions. The code generation is generally semi-automated for the realization of tool adapters, but can be fully automated in certain cases. The code generation
is fully automated for the rest of the tool chain. Code generation thus reduces the effort for implementing the tool chain.

To show the feasibility of the automated support for tool chain development, the above described methods for the different phases of tool chain development have been implemented and applied in case studies. The implementation is based on the systematic description of tool chains provided by the modeling language TIL.

9.2 Future Work

The presented results provide opportunities for continuing this work. Existing work can be extended by deepening unexplored paths spotted along the way and by reducing some of the delimitations (see Section 1.4). Some of the listed future work has already started.

9.2.1 Synthesis for Different Integration Frameworks

By design, TIL describes a tool chain independently of a particular implementation technology or tool integration framework. Thus the TIL approach could be used as a platform for generating tool chain implementations for different integration technologies. In this work a code generator for TIL is presented, which produces OSLC-compliant tool chains. Future work is needed to evaluate whether alternative code generator for TIL can be created for generating an implementation that uses an existing tool integration framework, such as ModelBus [85] or Jazz [78]. Generators targeting different integration frameworks might share commonalities, which can be reused. The reuse among generator parts can be described using variability management of the code generators.

9.2.2 Evolution of Tool Chains and Integrated Tools

Tool chains need to co-evolve with the tools they integrate. Such an evolution can occur if a new version of an integrated tool becomes available or if an integrated tool needs to be replaced by a tool with similar functionality. These changes may affect the internal part of the ToolAdapter but not necessarily require a change of the ToolAdapter metamodel, so the rest of the tool chains is not affected by the change. If the change requires an adaptation of the ToolAdapter metamodel, additional support for managing this change by migrating the tool data to the new tool may become necessary. Initial experiences with automatically migrating data between two similar tools have been explored, using a variant of the heuristic approach for generating model transformation described in Appended Paper G. Future work needs to extend and evaluate the approach and provide case studies for different migration scenarios.
9.2.3 Formal Semantics and Simulation

Initial work on formalizing the behavioral semantics of TIL has been performed [31] by defining a mapping from TIL models to networks of finite state machines (FSMs). The mapping is implemented as a model transformation, which can be used for the semantic anchoring [57] of TIL.

The tool chain architect may want to explore the dynamic behavior of the tool chain early in the development process, without code generation and deployment. Simulation can be used for early validation of the behavior of a tool chain described by a TIL model. The mapping from TIL to FSMs can be used for producing a simulation model of the behavior of a TIL model. UPPAAL [111] provides a graphical environment for creating and simulating networks of finite state machines, so the simulation model resulting from the model transformation can be interpreted by the UPPAAL simulator for step-by-step analysis of the tool chain behavior and to produce execution traces. Initial work on the simulation of tool chain behavior has been described [31].

9.2.4 Analysis of Additional Properties and Tradeoffs

Work on non-functional properties of tool chains, especially on the economic implications, has not received much attention from academia [164]. Section 6.5.3 explored the feasibility of analyzing the development cost and the cost savings achievable by tool chains based on the TIL model. The work on cost estimation for tool chains has an explorative character, focusing on the feasibility of automated support based on the TIL description. An in-depth evaluation of the results is part of future work.

Analysis of additional properties, such as scalability or performance, can complement the analysis support for tool chains. To analyze these properties, additional information is required besides the structural information available in the TIL model. This information can be provided by extending TIL with annotations for the analyzed property. Further work is needed to choose appropriate analysis methods for these properties and evaluate them. The analysis results can be used to make well-informed design choices. When there are several analysis methods available that target different properties, their analysis results need to be combined. Future work needs to investigate the trade-offs between different tool chain properties.

9.2.5 Qualification of Tool Chains

If tool chains are used to develop a safety critical system, the tools and the tool chain might need to be qualified according to safety regulations. Due to the safety-criticality of embedded systems, safety standards are in place, such as IEC 61508 [61], ISO26262 [77] for the automotive industry and DO-178C/DO-330 [129, 130] for the aviation industry. These standards put strong demands on the development
process regarding certification of the embedded system, and require qualification of the development tools. In addition, tool chains might need to be further qualified [13] by detecting potential hazards. An initial approach [12] for the qualification of tool chains with TIL has been proposed. Future work needs to investigate to what extent the hazards related to tool integration can be identified from an explicit model of a tool chain expressed in TIL.

9.2.6 Enhanced Support for Presentation Integration

Current support for modeling presentation integration with TIL focuses on the user interface (UI) for the complete tool chain. Future work needs to evaluate different strategies for presentation integration based on the user interfaces of existing tools. This might require the introduction of new language concepts and extensions to the code generator.

One strategy for presentation integration is the replacement of existing tool UIs with a unified UI for all involved tools. An example for this strategy are the OSLC-compliant concepts for Delegated User Interfaces [141], which are web pages that provide a simplified view of the data and services of the tool. While a novice user might appreciate the simplified and unified UI, an expert might be disturbed by the change to the accustomed environment. Another strategy for presentation integration is the extension of existing UIs. This is, however, only possible for tools that allow an extension of their UI. For the scope of web applications, this strategy has been realized by overlays through browser-based plugins by CORSET [68]. Since the ToolAdapters created by TIL are web application for desktop tools, synergies can be achieved by combining both approaches.

9.2.7 Enhanced Support for Data Management

Data management is currently either supported by a repository that is part of a development tool, such as offered by DOORS\(^1\), or by using an SCM system that is external to the tool, such as Subversion\(^2\). In an industrial context, tool data needs to be handled by software configuration management (SCM) systems or product lifecycle management (PLM) systems for mechanical products. In the future we would like to combine the tool integration provided by TIL with these approaches for data management.

TIL currently offers the Repository language concept, realizes data storage and versioning functionality. This language concept of TIL could be extended to reflect more functionality of data management systems. This would allow the tool chain architect to explicitly model the chosen strategy for data management in the TIL model.

Alternatively, data management could become a part of each ToolAdapter. This alternative builds on the idea that the tool data is managed and owned by the

\(^1\)http://www.ibm.com/software/awdtools/doors
\(^2\)http://subversion.tigris.org
9.2. **FUTURE WORK**

ToolAdapter. Thus data management would be transparent in the TIL model, and the code for realizing data management for each tool is created by the code generator as part of the implementation of each ToolAdapter.

As this example shows, there is a tradeoff between making concepts for tool integration explicit in the TIL model by dedicated language concepts and hiding the tool chain functionality by introducing the functionality in the generator. Future work needs to evaluate these -- and potentially additional -- approaches for both modeling and using data management in tool chains, including the implications for other parts of the tool chain, such as data-tracing and data-transfer.

### 9.2.8 Extended Evaluation of TIL

This work on tool integration was performed and evaluated in the context of embedded systems development, which has an extensive demand for tool integration, as the heterogeneity of the system requires a wide range of different development tools (see Section 1.2). Other areas of software and system development also face the challenge of tool integration to a certain extent. The work presented here may thus be applicable in the wider context of software and system development. Future work needs to evaluate the applicability and suitability of TIL in these new areas.

As a complement to the evaluation through case studies, a controlled experiment could be designed to compare the TIL approach to other approaches for developing tool chains, such as the completely manual (CM) implementation approach. Of interest is a comparison regarding the time necessary for tool chain development or the quality of the tool chain implementation. To implement such a controlled experiment, a number of engineers would be divided into two groups, where one group uses TIL, while the control group uses an alternative approach.

On the one hand, using a controlled experiment has the advantage that many relevant variables can be controlled. On the other hand, controlled experiments would need to take the tool chain into a laboratory setting and drastically limit the size of the developed tool chain, which conflicts with the need for a more realistic tool chain. In addition, there are a number of practical limitations, such as the availability of a sufficiently large number of engineers and their time commitment. Future work may thus include a controlled experiment for the development of a small tool chain with students as subjects.
Bibliography


Part IV

Appended Papers