Dissertation

Model Interoperability between Meta-Modeling Environments by using M3-Level-Based Bridges

Heiko Kern

University of Leipzig
Faculty of Mathematics and Computer Science
Institute of Computer Science

Leipzig, 24.08.2016
Contents

List of Figures VII
List of Tables IX

1 Introduction 1
   1.1 Interoperability between Meta-Modeling Tools . . . . . . . . . . . . 1
   1.2 Research Problem and Objective . . . . . . . . . . . . . . . . . . . 3
   1.3 Research Design . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
   1.4 Contributions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
   1.5 Outline of the Thesis . . . . . . . . . . . . . . . . . . . . . . . . . 6

2 Foundation in Modeling and Transformation 9
   2.1 Modeling . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
      2.1.1 Model-Driven Engineering . . . . . . . . . . . . . . . . . . . 9
      2.1.2 Models . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
      2.1.3 Modeling Languages . . . . . . . . . . . . . . . . . . . . . . 13
      2.1.4 Meta-Modeling . . . . . . . . . . . . . . . . . . . . . . . . . 14
   2.2 Model Transformation . . . . . . . . . . . . . . . . . . . . . . . . . 19
      2.2.1 Definition of Transformation . . . . . . . . . . . . . . . . . 19
      2.2.2 Characteristics of Transformations . . . . . . . . . . . . . . 20
      2.2.3 Transformation Paradigms . . . . . . . . . . . . . . . . . . . 22
      2.2.4 Epsilon Transformation Language . . . . . . . . . . . . . . 23
      2.2.5 Typical Usage of Transformations . . . . . . . . . . . . . . 24
   2.3 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25

3 Interoperability between Meta-Modeling Environments 27
   3.1 Definition of Model Interoperability . . . . . . . . . . . . . . . . . 27
   3.2 Structure of the Study . . . . . . . . . . . . . . . . . . . . . . . . . 28
      3.2.1 Objective of the Study . . . . . . . . . . . . . . . . . . . . . 28
      3.2.2 Scope of the Study . . . . . . . . . . . . . . . . . . . . . . . 28
      3.2.3 Tool Selection and Analysis . . . . . . . . . . . . . . . . . . 32
   3.3 Results of the Study . . . . . . . . . . . . . . . . . . . . . . . . . . 34
      3.3.1 Unification Mechanism . . . . . . . . . . . . . . . . . . . . . 34
      3.3.2 Model Level . . . . . . . . . . . . . . . . . . . . . . . . . . . 35
      3.3.3 Degree of Interoperability . . . . . . . . . . . . . . . . . . . 36
      3.3.4 Further Observations . . . . . . . . . . . . . . . . . . . . . . 37
      3.3.5 Threats to Validity . . . . . . . . . . . . . . . . . . . . . . . 38
      3.3.6 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38
### Contents

3.4 Interoperability in other Technologies ........................................ 39  
3.4.1 Technologies with a Three-Level Architecture ......................... 39  
3.4.2 Interoperability Approaches ........................................... 40  
3.5 Summary ............................................................................. 42  

4 M3-Level-Based Bridge ............................................................. 43  
4.1 Transformation Approach ..................................................... 43  
4.1.1 Requirements for Model Interoperability ............................... 43  
4.1.2 Transformation Description ............................................. 45  
4.1.3 Transformation Algorithm ............................................... 47  
4.1.4 Transformation Characteristics ..................................... 49  
4.2 Features of M3-Level-Based Bridges ...................................... 50  
4.2.1 Direction ....................................................................... 50  
4.2.2 Configuration ................................................................. 51  
4.2.3 Output .......................................................................... 52  
4.2.4 Validation ...................................................................... 52  
4.3 Development of M3-Level-Based Bridges .................................. 53  
4.3.1 Transformation System ................................................... 53  
4.3.2 Implementation Aspects ................................................ 54  
4.3.3 Development Process .................................................... 55  
4.4 Combination of M3-Level-Based Bridges .................................. 56  
4.4.1 Combination Approach ................................................... 56  
4.4.2 Alignment Transformation ............................................. 58  
4.5 Summary ............................................................................. 59  

5 Analysis and Mapping of Meta-Modeling Languages ....................... 61  
5.1 Structure of the Comparative Analysis .................................... 61  
5.2 Analysis of Meta-Modeling Environments ................................ 62  
5.2.1 Architecture of Integrated Information Systems .................... 62  
5.2.2 Cubetto Toolset ............................................................... 64  
5.2.3 Eclipse Modeling Framework ....................................... 66  
5.2.4 Generic Modeling Environment ..................................... 67  
5.2.5 MetaEdit+ ................................................................. 68  
5.2.6 Modeling SDK for Visual Studio .................................... 69  
5.2.7 Microsoft Visio ............................................................. 70  
5.3 Comparison of Meta-Modeling Concepts .................................. 72  
5.3.1 Basic Concepts .............................................................. 72  
5.3.2 Additional Concepts ..................................................... 78  
5.3.3 Further Observations .................................................... 80  
5.4 Mapping of Meta-Modeling Concepts ..................................... 82  
5.4.1 Basic Mappings ............................................................. 82  
5.4.2 Mapping of Relation Type ............................................. 83  
5.4.3 Attribute Mapping ........................................................ 89  
5.4.4 Mapping of Inheritance ............................................... 92  
5.4.5 Further Transformation Aspects ..................................... 92  
5.5 Summary ............................................................................. 94
6 Application of M3-Level-Based Bridges

6.1 Overview

6.2 Bridge 1: ARIS and Eclipse Modeling Framework

6.2.1 Bridge Architecture

6.2.2 M2-Transformation

6.2.3 M1-Transformation

6.2.4 Implementation

6.2.5 Example

6.3 Bridge 2: MetaEdit+ and Eclipse Modeling Framework

6.3.1 Bridge Architecture

6.3.2 M2-Transformation

6.3.3 M1-Transformation

6.3.4 Implementation

6.3.5 Example

6.4 Bridge 3: Microsoft Visio and Eclipse Modeling Framework

6.4.1 Bridge Architecture

6.4.2 M2-Transformation

6.4.3 M1-Transformation

6.4.4 Implementation

6.4.5 Example

6.5 Combination of MetaEdit-EMF and Visio-EMF Bridge

6.6 Summary

7 Evaluation of M3-Level-Based Bridges

7.1 Design of the Evaluation

7.2 Application of Metrics

7.2.1 Number of Rules

7.2.2 Lines of Code

7.2.3 Halstead Metric

7.2.4 Transformation Runtime

7.3 Evaluation Criteria

7.3.1 Completeness

7.3.2 Complexity

7.3.3 Effort

7.3.4 Reusability

7.3.5 Applicability

7.4 Summary

8 Conclusion and Future Work

8.1 Summary and Conclusion

8.1.1 Interoperability between Meta-Modeling Environments

8.1.2 Transformation of Models and Meta-Models

8.1.3 Understanding of Meta-Modeling Languages

8.2 Directions for Future Research

8.2.1 Interoperability between Meta-Modeling Environments
# List of Figures

1.1 Lack of interoperability between meta-modeling environments . . . . 2  
1.2 Structure of the doctoral thesis . . . . . . . . . . . . . . . . . . . . 7  
2.1 Model hierarchy . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15  
2.2 Model, meta-model and meta-metamodel . . . . . . . . . . . . . . . 16  
2.3 Meta-modeling and modeling in MetaEdit+ . . . . . . . . . . . . . . 18  
2.4 Model transformation . . . . . . . . . . . . . . . . . . . . . . . . . . 19  
3.1 Scope of the study about interoperability . . . . . . . . . . . . . . . 29  
3.2 Technologies with a three-level architecture . . . . . . . . . . . . . . 40  
3.3 Transformation approach in JAXB . . . . . . . . . . . . . . . . . . . 41  
4.1 Equality of source and target model . . . . . . . . . . . . . . . . . . 43  
4.2 M3-level-based bridge . . . . . . . . . . . . . . . . . . . . . . . . . . 45  
4.3 Transformation directions . . . . . . . . . . . . . . . . . . . . . . . . 51  
4.4 Configuration of meta-model transformations . . . . . . . . . . . . . . 51  
4.5 Transformation output . . . . . . . . . . . . . . . . . . . . . . . . . . 52  
4.6 Validation support . . . . . . . . . . . . . . . . . . . . . . . . . . . . 52  
4.7 Abstract M3B transformation system . . . . . . . . . . . . . . . . . . 53  
4.8 Combination of M3-level-based bridges . . . . . . . . . . . . . . . . . 57  
4.9 Abstract syntax of the mapping language . . . . . . . . . . . . . . . 58  
5.1 Meta-metamodel of ARIS . . . . . . . . . . . . . . . . . . . . . . . . 63  
5.2 Data structure for ARIS models . . . . . . . . . . . . . . . . . . . . . 64  
5.3 E3 – Meta-metamodel of Cubetto . . . . . . . . . . . . . . . . . . . . 65  
5.4 Ecore – Meta-metamodel of Eclipse Modeling Framework . . . . . . 66  
5.5 Meta-metamodel of Generic Modeling Environment . . . . . . . . . . 67  
5.6 GOPPRR – Meta-metamodel of MetaEdit+ . . . . . . . . . . . . . . . 69  
5.7 Meta-metamodel of Modeling SDK for Visual Studio . . . . . . . . . 70  
5.8 Meta-metamodel of Visio . . . . . . . . . . . . . . . . . . . . . . . . 71  
5.9 Data structure for Visio models . . . . . . . . . . . . . . . . . . . . . 71  
5.10 Basic meta-modeling concepts . . . . . . . . . . . . . . . . . . . . . 72  
5.11 Typical variants of relations . . . . . . . . . . . . . . . . . . . . . . . 76  
6.1 Bridge between ARIS and EMF . . . . . . . . . . . . . . . . . . . . 96  
6.2 Abstract ARIS meta-model in EMF . . . . . . . . . . . . . . . . . . . 99  
6.3 Transformation example of ARIS-EMF bridge . . . . . . . . . . . . . . 101  
6.4 Bridge between MetaEdit+ and EMF . . . . . . . . . . . . . . . . . . 102  
6.5 Abstract MetaEdit+ meta-model in EMF . . . . . . . . . . . . . . . . 105
| 6.6 | Transformation example of MetaEdit-EMF bridge | 107 |
| 6.7 | Bridge between Visio and EMF | 108 |
| 6.8 | Transformation example of Visio-EMF bridge | 112 |
| 6.9 | Combination of MetaEdit-EMF and Visio-EMF bridge | 113 |
| 6.10 | Example of an EPC meta-model alignment | 114 |
| 6.11 | Transformation example for EPC models | 116 |

| 7.1 | Number of rules | 123 |
| 7.2 | Lines of code | 126 |
| 7.3 | Halstead difficulty | 129 |
| 7.4 | Runtime | 132 |
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Methodological foundation</td>
<td>4</td>
</tr>
<tr>
<td>3.1</td>
<td>List of meta-modeling tools</td>
<td>33</td>
</tr>
<tr>
<td>3.2</td>
<td>Model exchange between meta-modeling tools</td>
<td>37</td>
</tr>
<tr>
<td>5.1</td>
<td>Basic meta-modeling concepts</td>
<td>73</td>
</tr>
<tr>
<td>5.2</td>
<td>Features of relation types</td>
<td>74</td>
</tr>
<tr>
<td>5.3</td>
<td>Support of relation variants</td>
<td>76</td>
</tr>
<tr>
<td>5.4</td>
<td>Features of attributes</td>
<td>78</td>
</tr>
<tr>
<td>5.5</td>
<td>Additional concepts</td>
<td>79</td>
</tr>
<tr>
<td>5.6</td>
<td>Mapping of basic concepts</td>
<td>83</td>
</tr>
<tr>
<td>5.7</td>
<td>Mapping of relation types: arity</td>
<td>85</td>
</tr>
<tr>
<td>5.8</td>
<td>Mapping of relation types: multiplicity</td>
<td>86</td>
</tr>
<tr>
<td>5.9</td>
<td>Mapping of relation types: composition</td>
<td>87</td>
</tr>
<tr>
<td>5.10</td>
<td>Mapping of relation types: object-set</td>
<td>88</td>
</tr>
<tr>
<td>5.11</td>
<td>Mapping of relation types: role</td>
<td>89</td>
</tr>
<tr>
<td>5.12</td>
<td>Mapping of attributes: multiplicity</td>
<td>90</td>
</tr>
<tr>
<td>5.13</td>
<td>Mapping of attributes: ordered</td>
<td>91</td>
</tr>
<tr>
<td>5.14</td>
<td>Mapping of attributes: unique</td>
<td>91</td>
</tr>
<tr>
<td>5.15</td>
<td>Mapping of inheritance</td>
<td>92</td>
</tr>
<tr>
<td>7.1</td>
<td>Number of rules for the ARIS-EMF bridge</td>
<td>121</td>
</tr>
<tr>
<td>7.2</td>
<td>Number of rules for simple model transformations in ARIS</td>
<td>122</td>
</tr>
<tr>
<td>7.3</td>
<td>Lines of code for the ARIS-EMF bridge</td>
<td>124</td>
</tr>
<tr>
<td>7.4</td>
<td>Lines of code for simple model transformations in ARIS</td>
<td>126</td>
</tr>
<tr>
<td>7.5</td>
<td>Halstead values for the ARIS-EMF bridge</td>
<td>128</td>
</tr>
<tr>
<td>7.6</td>
<td>Halstead values for a simple model transformation</td>
<td>128</td>
</tr>
<tr>
<td>7.7</td>
<td>Runtime of the ARIS-EMF bridge and a model transformation</td>
<td>131</td>
</tr>
<tr>
<td>A.1</td>
<td>Modeling tools</td>
<td>145</td>
</tr>
<tr>
<td>A.2</td>
<td>Import and export formats of meta-modeling tools</td>
<td>147</td>
</tr>
</tbody>
</table>
Abstract

Models and modeling languages play an important role in software development and many other engineering disciplines. Generally, a model is an abstract and preferably formal representation of a system. Models improve the understanding of systems and facilitate the communication between different stakeholders participating in the development process. Beyond that, modern software development approaches such as Model-Driven Software Development use models for automating the development through model transformation, code generation, model validation or model-based testing. Nowadays, modeling is supported by powerful modeling tools. An important requirement for modeling tools is the capability to exchange models with other tools. This exchange capability enables development of tool chains covering the entire software development processes. Furthermore, the exchange of models enables the replacement of tools by another one fitting better to a changed development process.

Despite a variety of existing approaches, the exchange of models is still a serious problem. This work deals with this problem and presents an approach enabling the migration of models and associated languages between different meta-modeling tools. Meta-modeling tools are a special class of modeling tools allowing the definition of languages and the use of these self-defined languages. The proposed approach is denoted as M3-level-based bridge (M3B). The central idea of an M3B is the application of two coupled model transformations. The first transformation is responsible for the conversion of modeling languages. Based on the first transformation, the second transformation implements the migration of models. This thesis describes the transformation approach itself and various development aspects. A central aspect of this approach is the mapping between different meta-modeling languages. Thus, this work investigates a set of meta-modeling languages, extracts typical meta-modeling concepts, unifies these concepts in a common framework, and suggests typical mapping rules between these concepts.

The M3B approach and the proposed mapping rules are applied to the development of bridges. The developed bridges are between ARIS and EMF, Microsoft Visio and EMF, and MetaEdit+ and EMF. The bridges show the general usability of this approach. Additionally, the bridges serve as a foundation for a detailed evaluation. The evaluation concerns criteria, such as applicability, completeness, complexity, effort and reusability. The evaluation results are generally positive and twofold. In comparison to a simple model transformation, the M3B approach necessitates a higher complexity during the development phase, yet the M3B approach is more efficient because the approach covers the exchange of arbitrary models in dependency of their meta-models. Thus, the M3B approach significantly improves the model interoperability between meta-modeling tools.
Kurzfassung


1 Introduction

1.1 Interoperability between Meta-Modeling Tools

Models are a fundamental concept to handle the development of complex systems in many engineering disciplines. Generally, a model is an abstract and preferably formal representation of a system [Stachowiak 1973]. Based on models, different operations can be applied in order to support the engineering process. A typical engineering discipline using models is the area of software development. In this area, models are mainly used in the design phase to describe software at a higher abstraction level than the concrete implementation. Thus, models improve the understanding of software systems and facilitate the communication between different stakeholders participating in the development process. Beyond that, modern software development approaches such as Model-Driven Software Development (MDSD) [Stahl and Völter 2006] or Domain-Specific Development (DSM) [Kelly and Tolvanen 2008] use models for automating the development through model transformation, code generation, model validation or model-based testing. Typical models in software engineering are, for instance, UML diagrams [Fowler 2003], Entity-Relationship diagrams [Chen 1976], BPMN models [OMG 2011] or models expressed by domain-specific languages [Kelly and Tolvanen 2008].

The creation of models is the result of a modeling process supported by a modeling tool. Although a theoretical foundation in the modeling area is indispensable, a suitable tool infrastructure is necessary to facilitate the practical usage. Modeling is a tool-centric approach and hardly imaginable without modeling tools. Current tools offer a variety of features which support the user during the modeling process. Usually, modeling tools offer a set of predefined modeling languages. In addition to these tools, there are meta-modeling tools which enable as special feature the definition of languages and the usage of these self-defined languages. Modeling concepts of these languages are specified in form of meta-models. The ability of meta-modeling tools to define new (domain-specific) languages is one of the main features. Additional to this, meta-modeling tools can offer model processing capabilities to define domain-specific operation on models. Examples of such meta-modeling tools are: Microsoft Visio [Biafore 2007], MetaEdit+ [Kelly and Tolvanen 2008] or the ARIS tool [Kern 2007a].

An important requirement for tools, including meta-modeling tools, is interoperability with other tools. Generally, interoperability enables the cooperation or inter-operation between tools. In the context of this work, interoperability deals with the ability to exchange models between two or more tools. These exchanged models must be usable in the participating tools. There are manifold motivations for interoperability. One possible scenario is the setup of a complex tool chain.
ten tools are specialized on a specific task in the development process. Tools have to work together or inter-operate in order to cover a complete process. Therefore, a successful application of the whole development process depends heavily on the degree of interoperability between tools. Beside the aspect of cooperation, another important incentive for interoperability is the evolution of a tool landscape. As the software industry constantly evolves, modeling tools also evolve and old tools are being replaced by new ones better fitting the customer needs. The reasons for deciding to replace a tool can be manifold and depends on the one hand of the customer needs and on the other hand of the offered tool features. For instance, Kirchner and Jung [2007] propose a framework for the evaluation of meta-modeling tools. This evaluation framework shows the difficulties in selecting a suitable tool. One reason for replacing a tool could be that a tool offers better features for the processing of models. A further reason may be the license terms of tools, particular the costs. In order to avoid the vendor lock-in effect, interoperability between tools is necessary to enable the reuse of existing models between tools from different vendors.

The example in Figure 1.1 illustrates a possible scenario with two meta-modeling tools. On the left hand side is Microsoft Visio and on the right hand side is the Eclipse Modeling Framework (EMF) [Steinberg et al. 2009]. Both tools support a three-level meta-modeling architecture. Visio is widely-used modeling tool in different modeling domains and has its strength in graphical modeling. In contrast to Visio, EMF is a software framework which is suitable for processing of models. Based on EMF, there are additional frameworks and various model processing tools for transformation, validation and code generation. The exchange or migration of models is necessary to combine the advantages of both tools. In this case, the processing of Visio models with EMF tools would lead to a benefit.
1.2 Research Problem and Objective

Generally, we focus on the issue of missing interoperability between meta-modeling tools. Despite a variety of already existing approaches, interoperability of meta-modeling is a challenging task. In this thesis, we research a transformation-based approach for the migration of models between different meta-modeling tools.

1.2 Research Problem and Objective

Generally, the problem space of this thesis is interoperability between different modeling tools. This problem space can be restricted by the following two constraints. Firstly, the considered tools are restricted to the special class of meta-modeling environments. A meta-modeling environment is characterized by a model hierarchy consisting of three levels. The top level is the meta-metamodel that defines a set of meta-models which in turn defines a set of models. Such environments can be realized in form of a tool or a framework. Secondly, the interoperability issue is restricted to the conversion or migration of models and meta-models from one meta-modeling tool to another meta-modeling tool. The migration should allow the inter-operation of tools or the reuse of models from one tool in another tool.

On the one hand, there is a need for interoperability between different meta-modeling tools. The scenario in Figure 1.1 in the previous section shows a motivating example for this need. On the other hand, there is a lack of interoperability between these tools. Despite many approaches and initiatives for tool interoperability, the current state of this field is unsatisfied and no common standard has been accepted for the exchange of models. The majority of meta-modeling tools often provides tool-specific import and export formats for models. Current solutions are unsatisfactory because proprietary formats cannot solve the problem of having different meta-model hierarchies implemented in different tools.

The thesis addresses this lack of interoperability between meta-modeling tools and presents a transformation-based approach for the migration of models and meta-models between different meta-modeling tools. We denote this approach as M3-level-based Bridge (M3B). The objective of this work is the development of the M3B approach. In more detail, the thesis concentrates on the formal description of this transformation approach, the research of possible transformation features, and the discussion of various development aspects. An important issue concentrates on the derivation of mappings between different meta-modeling languages. These mappings are the central point for the development of M3Bs. The last issue concerns the application and evaluation of the M3B concept. The application of the M3B approach should show the functionality and usability of this transformation concept. The evaluation should identify advantages, disadvantages and the limits of this approach.

1.3 Research Design

The description of the research design helps to understand the research procedure and the structure of this work. Additionally, the research design can ease the interpretation of the results. The description of the research design is structured
into two parts. The first part addresses the general research paradigm and the second part addresses the concrete procedure. The research design of this work follows the paradigm of design science [Hevner et al. 2004]. The design science paradigm is a basic approach for solving problems in engineering disciplines. The paradigm is established in information systems research and focuses on the creation and evaluation of artifacts in the context of information technologies (IT) to solve identified problems. IT artifacts can be, for instance, software, formal logic, algorithms, methods or tools. Hevner et al. [2004] define a guideline for design science. We use this guideline to structure the research design of this work. An overview of the design science guideline and the correlating research work of this dissertation is presented in the following Table 1.1.

The first point in the guideline for design science is problem relevance. Generally, interoperability between modeling tools is an important issue in practical as well as theoretical context. The scenario in Section 1.1 shows a good example to understand the motivation for the integration of tools. In this specific case, the combination of Visio and EMF should enable the transformation of Visio models with modern EMF technologies. This scenario was identified as a real problem in a nationally funded research project with the name “Orchestration and Validation

<table>
<thead>
<tr>
<th>Design science guideline</th>
<th>Dissertation</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Problem relevance</td>
<td>- Research projects</td>
</tr>
<tr>
<td></td>
<td>- Study about interoperability</td>
</tr>
<tr>
<td></td>
<td>- Use cases from industry</td>
</tr>
<tr>
<td>- Design as an artifact</td>
<td>- M3B transformation approach</td>
</tr>
<tr>
<td></td>
<td>- Comparison and mapping of meta-metamodels</td>
</tr>
<tr>
<td></td>
<td>- Prototypes</td>
</tr>
<tr>
<td>- Design evaluation</td>
<td>- Prototypes and use cases</td>
</tr>
<tr>
<td></td>
<td>- Evaluation</td>
</tr>
<tr>
<td>- Research rigor</td>
<td>- Formal description</td>
</tr>
<tr>
<td></td>
<td>- Usage of a transformation language</td>
</tr>
<tr>
<td></td>
<td>- Usage of metrics in the evaluation</td>
</tr>
<tr>
<td>- Communication of research</td>
<td>- Publications and presentations</td>
</tr>
<tr>
<td></td>
<td>- Workshops</td>
</tr>
<tr>
<td></td>
<td>- Prototypes available as download</td>
</tr>
<tr>
<td>- Research contributions</td>
<td>- Study about interoperability</td>
</tr>
<tr>
<td></td>
<td>- M3B approach</td>
</tr>
<tr>
<td></td>
<td>- Comparison and mapping of meta-metamodels</td>
</tr>
<tr>
<td></td>
<td>- Prototypes and applications</td>
</tr>
<tr>
<td></td>
<td>- Evaluation</td>
</tr>
</tbody>
</table>

Table 1.1: Methodological foundation
1.3 Research Design

of Integrated Information Systems” [Stein et al. 2008]. Despite many powerful approaches for tool interoperability, there is no satisfying solution for the exchange of models and meta-models between different meta-modeling environments. This work wants to present a suitable solution to improve the interoperability between meta-modeling tools.

The result of design science research is, by definition, a purposeful IT artifact created to address an important problem. This artifact must be described effectively to enable its implementation and application in an appropriate domain. In this work, the artifact under research is the M3B approach. Based on this artifact, we consider related problems, for instance, the description of common meta-modeling concepts and the mapping between these concepts to support the M3B development. The development of the M3B approach is iterative and comprises three iterations. Each iteration has a specific purpose and is combined with the development of a bridge. The first iteration provides a first applicability of the M3B approach and focuses on the transformation between ARIS and EMF. Afterwards, the second iteration sharpens the understanding and improves the formalization of the M3B approach. This iteration includes the development of an M3B between MetaEdit+ and EMF. The third iteration finishes the research process and should emphasize the evaluation aspect of the M3B approach. This iteration focuses on the integration of Microsoft Visio and EMF.

The utility, quality, and efficacy of an artifact must be rigorously demonstrated via well executed evaluation methods. The evaluation of designed artifacts must use established methodologies. Hevner et al. [2004] describe a set of evaluation methods. Out of these methods, we select methods fitting to this work. The first method is the case study approach. We apply the bridge approach in three use cases: ARIS-EMF, MetaEdit-EMF, and Visio-EMF. Additionally, we use each bridge in real-world scenarios. Based on these use cases, we discuss the usability of the bridge. Furthermore, we use an analytical approach and measure different metrics, such as complexity, lines of code, or runtime. These metrics address static and dynamic aspects. We conclude the evaluation with an argumentative discussion. This discussion is based on the results of the measured metrics, the experiences made from the case studies, and the conceptual description of the M3B approach.

The last important aspect of this research design is the communication of research. Design science research must be presented effectively to technology-oriented as well as management-oriented audiences. This work and the results of this research are discussed mainly in the research community on different workshops with experts in this area. Furthermore, the results are published in articles at conferences. The implementations of two bridges are available as open source projects. The publication of articles and bridges yields feedback from researchers and users applying the bridges in real-world scenarios. Please see Appendix D for further details of the research communication aspect.
1.4 Contributions

This work presents a transformation-based approach allowing the migration of models between meta-modeling tools. A special feature of this approach is that every model can be transformed on the basis of a prior-executed meta-model transformation.

The first contribution is the transformation approach itself. We give a formal description of the transformation approach and derive possible features of this approach. Furthermore, we research applications of this approach and evaluate these applications to show the pros and cons. Regarding the field of model-to-model transformation, this thesis makes a contribution in the field of higher-order transformations. Regarding the interoperability issue, this work presents a serious solution for model migration between modeling tools.

The second contribution concerns the understanding of meta-modeling languages. This work analyzes and compares a set of meta-modeling languages. The comparison extracts typical meta-modeling concepts and creates a unified concept framework. This framework is used to define mapping rules between meta-modeling concepts. The rules can be regarded as knowledge which can be applied to further meta-modeling languages.

The third contribution is the implementation of concrete M3Bs. Precisely, we develop three M3B transformations between ARIS and EMF, Visio and EMF, and MetaEdit+ and EMF. Referring to these tools, we increase the interoperability between these tools and provide applicable solutions. Furthermore, we develop an approach and implement a corresponding tool which facilitates the combination of different bridges.

1.5 Outline of the Thesis

The structure of this thesis is presented in Figure 1.2. After the introduction, we define basic concepts in the field of modeling and transformation in Chapter 2. The foundation of both concepts uses the research field of Model-Driven Engineering.

Chapter 3 covers interoperability between meta-modeling tools. We define terms such as integration, interoperability, and model migration. After that, we present a study about interoperability between meta-modeling tools. The study shows current approaches for the exchange of models and measures the current degree of interoperability between a set of selected meta-modeling tools.

In Chapter 4, we present the M3B approach. We give a formal description of this approach, describe a corresponding transformation algorithm and present possible M3B features. We propose an abstract transformation system and discuss further development aspects. Furthermore, we present an approach for the combination of M3Bs.

The M3B approach is based on a mapping between meta-modeling concepts. For this reason, we investigate different meta-modeling languages in Chapter 5. We compare these languages and define a common framework with meta-modeling concepts and their variabilities. This framework is like a reference framework for
meta-modeling languages. Based on this framework, we define mappings rules and discuss advantages and disadvantage of these rules.

In Chapter 6, we apply the M3B approach and develop three M3Bs. The first bridge is between ARIS and EMF, the second bridge is between MetaEdit+ and EMF, and the third bridge is between Microsoft Visio and EMF. Each application includes an overview, a description of mapping rules, a description of the bridge implementation, and an example use case. Furthermore, we present the combination of the MetaEdit-EMF bridge and Visio-EMF bridge.

In Chapter 7, we evaluate the M3B approach. The evaluation is divided into two parts. The first part is the measurement of selected metrics, such as number of rules or lines of code. The second part covers the evaluation of selected criteria, such as applicability, complexity or effort. The evaluation is based on the implemented bridge prototypes and the results of the measured metrics.

Finally, we present a summary and conclusion of this work in Chapter 8 and discuss further research questions in this field.
2 Foundation in Modeling and Transformation

In this chapter, we introduce the theoretical framework of this work. The framework refers to the field of Model-Driven Engineering. This field is suitable to tackle the addressed problem of model transformations between meta-modeling environments.

2.1 Modeling

2.1.1 Model-Driven Engineering

Model-Driven Engineering (MDE) is the application of engineering with the focus on modeling and transformation in order to give a systematic and methodological support for software and system development. The objective of MDE is the identification, definition, formalization and development of concepts, methods and tools in a structured and scientific way [Schmidt 2006]. In this environment, there are a variety of methods, tools, conferences, journals, and books.

A cornerstone of MDE is Model-Driven Software Development (MDSD) [Stahl and Völter 2006] which can be regarded as an instance or realization of MDE. In other words, MDE can be considered as a generalization of MDSD [Favre 2005; Bézivin 2006]. MDSD emphasizes the application character and is applied in an industrial and practical context. In the following, we describe some well-known MDSD approaches.

Model-Driven Architecture (MDA) is a software design approach launched and standardized by the Object Management Group in 2001 [OMG 2014a]. MDA allows the software specification at different abstraction levels. These levels are denoted as Computation Independent Model (CIM), Platform Independent Model (PIM) and Platform Specific Model (PSM). The aim is a gradual refinement of models from CIM to code level. Additional to the level architecture, MDA provides different technologies. An important component is the Meta Object Facility (MOF) [OMG 2015b] which serves as metamodeling approach and allows the definition of modeling languages, such as UML or CWM [OMG 2003]. For model transformations, MDA propagates the Query View Transformation (QVT) language [OMG 2015a].

Architecture-Centric MDSD concerns the development of generative software architectures for software system families [Stahl and Völter 2006]. The approach differentiates between individual, schematic, and generic implementation components. The generic code is equal for all applications and can
be realized as runtime components or libraries. Schematic code components are structurally identical and can be produced on the basis of application-specific domain models and appropriate transformations. Individual code components are application-specific parts and should be implemented manually for each application system.

**Domain-Specific Modeling (DSM)** is a software development approach using domain-specific languages and code generators [Kelly and Tolvanen 2008]. A domain-specific language covers concepts of a defined domain or problem space and supports a compact expression of models. Based on languages and corresponding models, code generators can automatically create executable code.

**Model-Integrated Computing (MIC)** is an approach for Model-Driven Software Development in the field of electrical engineering, particular in the context of distributed real-time and embedded systems [Sztipanovits and Karsai 1997]. MIC uses models as central element in the life-cycle of systems including specification, design, development, verification, integration, and maintenance. During the design phase, MIC concerns the formal representation, composition, analysis, and manipulation of models. The approach is supported by the tool Generic Model Environment (GME). This tool allows the definition of domain-specific languages and facilitates the processing of models by an integrated framework.

**Generative Software Development (GSD)** addresses the creation of software system families [Czarnecki and Eisenecker 2000]. The development process consists of two parts: domain engineering and application engineering. The first part concerns the development of reusable assets such as components, generators, or domain-specific languages. The second part concentrates on the development of concrete applications by using reusable assets resulting from the domain engineering part. A further key concept in GSD is the generative domain model consisting of problem space, solution space and a mapping between both spaces. The problem space is a set of domain-specific abstractions that can be used to specify desired system-family members. The solution space consists of implementation-oriented abstractions which can be instantiated to create implementations. A mapping includes relations between a problem and solution space.

All these approaches share the same basic idea with some small differences. Generally, MDSD approaches use models as central assets in a development process. Models describe a software system in an abstract way. Based on models, transformations and code generators create source code and close the abstraction gap between models and executable code. Additional to the transformation operation, there are further operations supporting the development process (e.g. validation, weaving, or simulation). The goals of MDSD are, for instance, increase of development speed, enhancement of software quality and reusability, advancement of
complexity management, and improvement of communication between stakeholders involved in the development process.

Looking back on the history of software development, MDSD is the next logical step in the evolution of software development. Since the beginning of programming, there is a need to find appropriate abstractions. These abstractions can facilitate the handling of complexity during the software development process. Considering the aspect of language paradigms, we can observe that the first programs were written in machine languages followed by assembler languages. Afterwards, procedural and object-oriented languages were used for programming. All language paradigms abstract from the concrete execution at machine level. Modeling languages have the same idea and try to go one step further by using language concepts occurring in the problem domain of a user or application. Besides the concept of modeling, the concept of transformation was already used in order to close the abstraction gap. For instance, there is a variety of compilers transforming high-level languages into assembler code. Both concepts, languages and transformations, are used in existing software development approaches. However, MDSD wants to facilitate the definition of modeling languages, transformations, code generators and further components to support the software development process.

2.1.2 Models

Modeling is a fundamental approach to describe a system in an abstract representation. The system concept is used in various sciences such as natural sciences, computer science, or electrical engineering. A general view on the system concept is part of the General Systems Theory [von Bertalanffy 1976]. Generally, a system is a set of interacting and interdependent components forming an integrated whole. In more detail, systems are defined by the following characteristics: systems have a structure defined by components or elements and their composition, systems have behavior, and systems have inter-connectivity, that is, various parts of a system have functional and structural relationships to each other. Thus, software can be regarded as a complex system composed of different entities with an implemented behavior.

The process of modeling results in a model. Stachowiak [1973] describes in his work Allgemeine Modelltheorie three characteristics of a model defining this concept in a general sense.

**Representation** A model represents a system. The representation relationship between model and system should be a homomorphism. The system can already exist or will be created based on the model.

**Abstraction** A model suppresses irrelevant details and focuses on important aspects. Typical abstractions are reduction, generalization or classification.

**Pragmatics** Every model is created for a certain purpose and has a pragmatic trait. Pragmatic depends on the context of a model. The context is determined by a viewer or user of the model, time of creation and usage, and objective of the model.
The idea behind modeling is to have an abstract description of a system. Modeling is an important approach to handle complexity of systems in the real world. Models can be used for a demonstration or visualization of complex systems. Models improve the understanding of a system and allow the derivation of new knowledge. Furthermore, models can also serve as a plan or a template for building a system and a model can be used as a replacement for a system. If the direct application of operations is impossible or too expensive, a substitution allows the execution of operations on a model instead of the system.

The process of abstraction refers to an action of mind in order to isolate certain characteristics. Important properties are highlighted and unimportant properties are omitted during the abstraction. The decision whether a property is important or unimportant depends on pragmatic reasons and varies with the interest of awareness [Prechtl and Burkard 1999]. Generally, we can regard the process of modeling as the function $modeling$ that produces a model $M$ from a system $S$:

$$M = modeling(S).$$

(2.1)

Corresponding to the characteristics of models from Stachowiak [1973], the modeling function consists of an abstraction feature and a representation feature. The representation feature can be divided again in a projection and a translation. This differentiation leads to the following modeling function from Kühne [2006]:

$$modeling = abstraction \circ projection \circ translation.$$  

(2.2)

Projection is a structure preserving operation or injective homomorphism creating a relationship between a model $M$ and (a part of) the original system $S$ [Kühne 2006]. Important for the projection is analogical thinking in order to find a useful mapping. The translation changes the syntactical representation of the original system $S$ and requires a formalization by a modeling language.

Besides the projection and translation, there are further important forms of abstraction such as generalization or classification. Classification is a systematic assortment of similar concepts, objects, and phenomena into groups [Baumann et al. 2000]. A large number of individual instances or objects are grouped into a new entity which is often denoted as class. Generally, all objects of a class have the same set of properties but can have different property values. Generalization allows the deduction of more universal statements. Generalization decreases the number of class-specific criteria and increases the number of class members (objects).

The usage of models in the context of computer science is manifold [Thomas 2002]. In the area of software development, models are mainly used during the design phase to specificity or describe the structure and behavior of a software system. There are a lot of modeling languages covering a certain software domain and different aspects. Known languages are, for instance, Entity-Relationship-Diagrams for relational data models or the Unified Modeling Language for object-oriented systems. Based on these models, the implementation can be derived manually or automatically. The latter option leads to the automation of the development process which is the idea of Model-Driven Software Development.
2.1 Modeling Languages

Generally, languages enable the communication between different stakeholders. A language defines a vocabulary, rules for building sentences, and the meaning of the vocabulary and possible combinations [Gove 1993]. We can distinguish between natural and artificial languages. A natural language is spoken by persons and is the result of a diachronic development. Artificial languages have been developed by humans for various reasons and for different purposes. The set of atomic expressions and rules for the combination of atomic expressions can be formally described through mathematics or logic. Depending on the syntactical look and the application domain of a language, an instance of a language is called: sentence, program, formula, diagram or model. In the last two cases, a language can be denoted as a modeling language.

A modeling language $L$ is an artificial language which is formally defined as a 5-tuple consisting of concrete syntax $C$, abstract syntax $A$, semantic domain $S$, semantic mapping $m_s : A \rightarrow S$ and syntactic mapping $m_c : A \rightarrow C$ [Karsai et al. 2003]

$$L = (C, A, S, m_s, m_c).$$

Syntax determines the structure of a language and semantics defines the meaning of language constructs and their combination. Furthermore, syntax comprises the abstract $A$ and concrete $C$ syntax. The semantic domain $S$ defines concepts or domain entities in a certain domain. A semantic mapping $m_s$ defines a relation between concepts of the abstract syntax $A$ and concepts of the semantic domain $S$. A syntactic mapping $m_c$ defines a relation between elements of the abstract syntax $A$ and elements of the concrete syntax $C$.

Concrete Syntax

Concrete syntax of a modeling language refers to a specific human-usable representation of modeling elements. In principle, we can distinguish between textual and graphical modeling languages. A textual language consists of characters leading to a linear character string. These characters have no pictorial character. The concrete syntax of a graphical language is defined by geometric shapes such as lines, arrows, closed curves, rectangles, boxes, icons or symbols. These geometric shapes can contain textual elements. Principles such as linking, nesting or partitioning can be used for a topological structure of models [Harel and Rumpe 2004]. Graphical modeling languages are also called visual formalisms or visual languages. In contrast to textual languages, graphical modeling languages have no standard description for the concrete syntax. But there are proprietary approaches such as gmfgraph as part of the Graphical Modeling Framework (GMF) [Gronback 2009] or the Standard Vector Graphics (SVG) [W3C 2011].

Abstract Syntax

Abstract syntax defines concepts of a language (e.g. entities and relationships between these entities) and possible compositions of modeling elements [Karsai et al.
In other words, abstract syntax defines the structure of a language [Stahl and Völter 2006; Greenfield et al. 2004; Mellor et al. 2004; Frankel 2003]. Analogous to the definition of a concrete syntax, there are different approaches to define an abstract syntax. The abstract syntax for a textual language groups words into expressions or sentences [Harel and Rumpe 2004]. A standard approach to define a textual language is the usage of formal grammars specifying a set of production rules. A set of characters are a valid expression or sentence of a language, if and only if the grammar produces this expression [Hopcroft et al. 2006]. There are two methods for the definition of graphical modeling languages: graph grammars or meta-modeling. The idea of graph grammars is that models are interpreted as graphs. Often these graphs are directed, typed and labeled graphs. A graph grammar applies and extends the concept of grammars to graphs [Ehrig et al. 2006].

The meta-modeling approach defines language concepts and their relationships by using models which have often an Entity-Relationship-like description. These models are denoted as meta-models. In comparison to the meta-modeling approach, graph grammars have a better mathematical foundation. Due to the direct representation of language concepts, the meta-modeling approach is more intuitive in the definition of the abstract syntax [Harel and Rumpe 2004]. This is one reason for their prevalence in the modeling domain [Greenfield et al. 2004].

2.1.4 Meta-Modeling

Meta-Modeling and Model Hierarchy

The Greek prefix “meta” can be translated as “after” and expresses that an operation is executed twice in succession [Gove 1993]. This means, meta-modeling is a modeling activity applied to modeling itself. Additionally, the prefix intends an order between participating things, that is, the meta-thing is on a higher level than the (normal) thing. Generally, meta-modeling can be regarded as a process defining the process of modeling. For this purpose, meta-modeling addresses the definition of various assets or artifacts concerning a modeling process. These assets are, for instance, “logic, reasoning processes, guarantees of truth, proofs, and axioms of validity” [van Gigch 1991, 92]). Strahringer [1996] confirms this statement and differentiates between two main aspects of meta-modeling. The first aspect concerns the process of modeling as a methodology and the second aspect concerns the concepts of a language. However, meta-modeling is often focused on the language aspect and refers to the construction and development of a modeling language, particularly the definition of syntax and semantics. This work also deals with the language definition aspect and ignores the methodology aspect.

A meta-model is a model of models [OMG 2014a]. This statement is ambiguous and requires a more detailed description of the relation between meta-model and model. A model of models is equal to an application of the previously defined modeling function in Equation 2.1. This application results in a meta-model. The input of this modeling function is the set of all possible models. Furthermore, the modeling function requires as abstraction operation \( \alpha \) (see Equation 2.2) the usage of a classification operation. Thus, a meta-model consists of types or meta-model...
elements defining a possible set of elements in a model. The classification operation being an abstraction increases the abstraction level of a meta-model in comparison to a model. The relation between meta-model and models can be compared to the type-token distinction in the area of knowledge representation [Wimmer 2005]. The distinction between type and token is a separation between types in a meta-model and tokens in a model. Alternative names of this relationship between models and meta-models are “conform to”, “linguistic instance of” [Atkinson and Kühne 2003], “is captured in” [Mellor et al. 2004] or “is instance of” [OMG 2014a]. We define the conform to relation as a function which maps a model $M$ into a meta-model $MM$:

$$\chi : M \mapsto MM. \quad (2.3)$$

A meta-model defines the abstract syntax of a modeling language [Mellor et al. 2004; Stahl and Völter 2006; Greenfield et al. 2004]. The relation between a meta-model $MM$ and an abstract syntax $A$ of a language $L$ is defined as the following function:

$$\sigma : MM \mapsto A. \quad (2.4)$$

The process of meta-modeling results in a meta-model defining the abstract syntax of a language. The definition of a meta-model requires a further language that is denoted as meta-modeling language. Analogous to a modeling language, the abstract syntax of a meta-modeling language is in turn defined by a meta-model. This meta-model can be denoted as meta-metamodel. Analogous to the relation between model and meta-model, the conform to relation can be applied between meta-model and meta-metamodel.

![Figure 2.1: Model hierarchy](image-url)
The relation between model, meta-model, meta-metamodel, language and meta-language is illustrated in Figure 2.1 (previous page). Model, meta-model, and meta-metamodel build a three-level model hierarchy. The bottom level contains models (M1), the level above includes meta-models (M2) and at the top level is a meta-metamodel (M3). Each model hierarchy possesses exactly one meta-metamodel that defines a set of possible meta-models. Each meta-model defines in turn a set of possible models. In some cases, a meta-metamodel is self-defined, that is, a meta-metamodel is described with the meta-metamodel itself. An example of this self-description is Ecore [Steinberg et al. 2009]. Strahringer [1996] and Kühne [2006] discuss further characteristics of model hierarchies. However, the classical three-level hierarchy is sufficient in the context of this work.

Figure 2.2 shows an example of a model hierarchy addressed in this work. On the left-hand side is an Event-driven Process Chain (EPC) describing a simple process. In the middle is a meta-model defining the core concepts of this EPC language. The syntax of the EPC meta-model is defined by a meta-modeling language. The meta-metamodel is on the right-hand side in Figure 2.2. The meta-metamodel comprises the concepts: model type, object type, and relation type. On the meta-model level, the rectangle with the name EPC defines a model type. The EPC model type consists of the object type Control flow node and the relationship type Control flow edge. The Control flow edge has two properties source and target defining the domain and co-domain of this relationship type. Control flow edge is an abstract type specialized by the object types: Event, XOR, AND, OR, and Function. The specialization of Control flow node allows to have a Control flow edge between all specialized nodes. The concrete syntax of the EPC language consists of the following elements and relates as follows to the meta-model: red hexagon → Event, green rounded rectangle → Function, circle with different signs → Connectors and arrow → Control flow edge.
Meta-Modeling Approach

Generally, there are two mechanisms for the definition of meta-models [Frankel 2003, 145]. The first variant is the heavyweight approach. This approach creates a meta-model from scratch. A language engineer uses a meta-modeling language and defines a completely new meta-model. Examples of this approach are the Meta Object Facility (MOF) or MetaEdit+. The second variant is the lightweight approach. This variant adapts or extends an already existing meta-model with domain-specific concepts. Examples for this approach are the stereotype mechanism from UML [Fowler 2003] or the ARIS filter concept [Kern 2007a]. Besides the definition mechanism, the specification mechanism of meta-models is an important aspect. We can distinguish between a visual specification, a dialog-oriented specification, or a textual description of meta-models.

Meta-Modeling Environment

A meta-modeling environment is an implementation of a model hierarchy. The implementation can be realized as a tool or framework. A meta-modeling environment enables the definition of modeling languages with the help of a meta-modeling approach. The environment facilitates modeling and storage of models conforming to a previously defined language. Additionally to the modeling capability, there are possibilities for model processing, such as transformations, code generators, or model validations. Synonyms for meta-modeling environment are meta-modeling tool, meta-modeling framework, meta-modeling platform [Karagiannis and Kühn 2002], domain-specific modeling tool [Kelly and Tolvanen 2008], meta-case tool [Kelly 1997], domain-specific language workbench [Ludwig and Salger 2006] or simply modeling tool. Generally, a meta-modeling environment comprises the following components.

Language workbench A language workbench enables the definition of a modeling language. A language engineer or designer can define the abstract and concrete syntax of a language. Depending on the workbench, there are dialog-based, textual or graphical approaches for describing a language. A workbench produces a language specification as development artifacts.

Modeling tool A modeling tool allows the creation of models conforming to a previously defined language. A modeling tool reads a language definition and sets up the modeling capability. There are two approaches for the set-up of a modeling tool. The first approach is the configuration of a generic modeling tool during runtime. The second approach uses generators to create code implementing a modeling tool. This generated code can extend an already existing framework. A modeling tool is used by a user or modeler.

Model storage A storage component is responsible for the persistence of models and meta-models. Often the storage implements the conform to relationship between models and meta-models and must ensure that all models conform
to a corresponding meta-model. Furthermore, a storage can provide additional features, such as, version control of models or multi-user support for cooperative modeling. The implementation can use simple files or complex databases.

**Model processing** An important aspect in a meta-modeling environment is the processing of models. Often, there are components or tools allowing the processing of models. Some environments offer the processing of models by a general programming language, other environments provide special transformation or generator languages.

**Interfaces** Interfaces enable the import and export of models and meta-models. This feature allows the integration of a modeling tool into a tool chain. There are interfaces on the data, function, and presentation layers. Interfaces on the data layer are often implemented as simple files. The function layer offers an API and the presentation layer enables the export of models as images.

An example of a meta-modeling environment is MetaEdit+. MetaEdit+ allows the definition of domain-specific modeling languages. The creation of meta-models (abstract syntax) is assisted by a dialog-based approach. The concrete syntax of a language can be created in a graphical editor. A generic modeling editor interprets the language specification and creates a specific modeling editor adapted to this language. Besides the modeling capability, MetaEdit+ offers a powerful model repository and a sophisticated processing possibility for models. Models can be processed with a generator language named MERL. Furthermore, interfaces enable the export and import of models and language specifications. MetaEdit+ provides an API and an XML-based import and export format. Figure 2.3 shows

Figure 2.3: Meta-modeling and modeling in MetaEdit+
the language workbench on the left-hand side and the modeling editor on the right-hand side. In this case, the workbench defines the EPC language. This language definition is depicted in the left dialog. The language consists of the graph type \textit{EPC} which includes the object types \textit{AND}, \textit{OR}, \textit{XOR}, \textit{Event}, \textit{Function}, and the relationship type \textit{Arc} with two role types \textit{From} and \textit{To}. The dialog in the middle shows the definition of the object type \textit{Event} and the dialog on the right shows the corresponding symbol for an event. The modeling tool on the right side shows the EPC language elements in a toolbar and an EPC model as example.

2.2 Model Transformation

2.2.1 Definition of Transformation

Generally, a model transformation defines a relationship between models by using a defined mapping between these models. Figure 2.4 shows the concept of a model transformation. Similar to a computer program, a transformation reads models, executes operations or applies defined rules, and creates corresponding output models. The input and output of a model transformation can be denoted as source and target model, respectively.

A transformation can be distinguished between a transformation definition and a transformation instance. A transformation definition describes relations between source and target models by using corresponding meta-models. Depending on the transformation system, a transformation definition is expressed by a transformation language. Typically, a transformation definition is an artifact at design-time. A definition comprises a set of rules describing the mapping of meta-model elements. In contrast to this, a transformation instance is an artifact at run-time and results from the execution or interpretation of a transformation definition. A transformation instance comprises a set of relations between model elements.

![Figure 2.4: Model transformation](image-url)
A transformation can be regarded as a function $\tau$ between a set of source models $M_s$ and a set of target models $M_t$. These models conform to a source meta-model $MM_s$ and a target meta-model $MM_t$, respectively. We define a transformation as:

$$\tau : M_s \rightarrow M_t, m_s \mapsto m_t \text{ where } \chi(m_s) = mm_s \in MM_s$$

$$\text{and } \chi(m_t) = mm_t \in MM_t.$$  

(2.5)

2.2.2 Characteristics of Transformations

Transformations have different properties. These properties relate to various aspects, such as development paradigm, execution of transformations, or input and output artifacts [Kern et al. 2006]. We describe selected properties necessary for the understanding of the M3B approach.

Model-to-Model and Model-to-Text Transformation

This property concerns the creation of target models. A model-to-model transformation ensures the conform to relationship between model and meta-model of the source as well as target side. That is, a transformation system reads a model and creates a target model conforming to a meta-model. During design-time, the system can use both meta-models to support the definition of valid transformation rules. At run-time, the referenced target meta-model is used to instantiate valid target models.

In contrast to a model-to-model transformation, a model-to-text transformation holds no explicit reference to a target meta-model. Depending on a source model, the transformation produces text which represents a model but the transformation system is not responsible for the creation of a valid target model. A model-to-text transformation is often implemented as a code generator [Czarnecki and Mens 2003].

Homogeneous versus Heterogeneous Transformation

A transformation $\tau : M_s \rightarrow M_t$ between source model $M_s$ and target model $M_t$ is homogeneous if the corresponding source meta-model $MM_s$ and target meta-model $MM_t$ are equal:

$$MM_s = MM_t.$$  

Otherwise, if the source and target meta-model are unequal: $MM_s \neq MM_t$, then the transformation is a heterogeneous transformation.

Endogenous versus Exogenous Transformation

Depending on a meta-metamodel, a transformation can be divided into endogenous and exogenous transformations. A transformation $\tau : M_s \rightarrow M_t$, whose corresponding source meta-model $MM_s$ and target meta-model $MM_t$ are instances
of the meta-metamodel $MMM_s$ and $MMM_t$, called endogenous transformation, if both meta-metamodels are equal:

$$MMM_s = MMM_t.$$ 

Otherwise, a transformation between models is denoted as exogenous transformations, if corresponding meta-models are instances of two different meta-metamodels, that is: $MMM_s \neq MMM_t$.

**Unidirectional versus Bidirectional Transformation**

Transformations have a direction which can be unidirectional or bidirectional [Czarnecki and Mens 2003]. A unidirectional transformation can be executed in one direction only. The transformation definition describes a relation from source to target. The transformation execution computes a target model based on a source model. A bidirectional transformation can be executed in both directions. A bidirectional transformation reads a source model and creates a target model. Additional to this case, the same transformation allows an execution in the inverse direction. A bidirectional transformation uses bidirectional rules or two separate complementary unidirectional rules, one for each direction. The invertibility of a transformation depends not only on the transformation rules, but also on the scheduling logic of the transformation system.

**Horizontal versus Vertical Transformation**

This characteristic concerns the abstraction level of input and output. A horizontal transformation processes source and target models which are at the same abstraction level. In contrast to this, a vertical transformation converts models which have a different abstraction level [Mens et al. 2006]. There is no method to measure the abstraction level of models but there are some indications to recognize the abstraction level. For instance, if the participating meta-models are equal, then the instantiated models are often on the same abstraction level. In the MDA context, the distinction between platform-independent (PIM) and platform-specific models (PSM) indicates the abstraction level. A PIM abstracts from implementation details and is closer to domain concepts. In contrast to this, a PSM is closer to a technology. In comparison, a PSM is on a higher abstraction level than a PIM. A transformation between a PSM and PIM is a vertical transformation. Often, the execution level is a fix point in order to determine the abstraction level. The execution level often has the lowest abstraction degree.

**Higher-Order Transformation**

A higher-order transformation (HOT) operates on model transformations, analogous to a simple transformation operating on models. A higher-order transformation has as input a transformation model and/or has as output a transformation model [Tisi et al. 2009, 20]. A transformation model represents a transformation
definition or a transformation instance. Tisi et al. [2009] describe typical tasks for higher-order transformations.

**Transformation synthesis** A transformation synthesis produces as output a transformation model. The input can be transformation models or other sources. An example of a synthesis is a transformation of an abstract mapping into an executable transformation (e.g. [Kern et al. 2014]).

**Transformation analysis** A transformation analysis has as input a transformation and creates as output different kind of data. The goal is the analysis of different aspects, such as validation of transformation rules, quality aspects, analysis of metrics (e.g. [van Amstel and van den Brand 2011]). The result of an analysis can be represented as a model again.

**Transformation (de)composition** A composition combines different input transformations to an output transformation. There are two methods to compose model transformations. An external composition connects separated and encapsulated transformations to a transformation chain (e.g. [Vanhooff et al. 2006]). An internal composition composes two transformations into another transformation. In this case, the granularity level is different. The internal composition is more fine-grained and works with information inside a model transformation (e.g. rules or modules). An example of internal composition is superimposition merging two transformation modules into a single transformation (e.g. [Wagelaar et al. 2010]).

**Transformation modification** The modification of transformations can be manifold. The creation of transformation variants is a typical scenario of a modification. Another scenario is feature weaving. HOTs can be easily used to weave cross-cutting concerns (e.g. debugging or traceability) into a model transformation.

### 2.2.3 Transformation Paradigms

Similar to programming paradigms, there are different paradigms to implement model transformations. Generally, we can differentiate between imperative and declarative approaches [Czarnecki and Mens 2003; Mens and van Gorp 2006]. The imperative approach allows the definition of command sequences and their control flow. Imperative transformation languages are similar to those for imperative programming languages. The imperative approach offers a high degree of control to a programmer. The transformation is described as a sequence of actions. This kind of description is useful if the order of transformation rules needs to be controlled explicitly.

In contrast to imperative transformations, declarative approaches express the logic of a transformation without describing its control flow explicitly. The focus of declarative approaches lies on what should be mapped by a transformation [Mens and van Gorp 2006]. Declarative model transformations describe a relationship
between source and target meta-models. The transformation system is responsible for correct rule application. Declarative transformations are in general more compact than iterative transformations because the declarative description is more concise. But, declarative approaches only have a limited applicability, since they are not suitable for complex calculations [Kolovos et al. 2015].

Declarative approaches can be differentiated into functional, relational, graph-based and template-based transformations. Functional approaches define transformations with the help of functions (e.g. XTend [Efftinge et al. 2008]). Relational approaches use relations (e.g. MTF [Demathieu et al. 2005]). The mathematical foundation of graph-based approaches (e.g. GREAT [Agrawal et al. 2003] or AGG [Ermel et al. 1999]) are graph grammars [Ehrig et al. 2006]. Models are interpreted as graphs, and graph transformations manipulate sub-graphs. Graph grammar rules use the visual syntax of source and target meta-models. Template-based approaches (e.g. XPand [Efftinge et al. 2008] or Java Emitter Templates (JET) [Popma 2004]) are associated with model-to-text transformations. A rule is called a template. The left side of a rule describes the matching condition and the right side describes the resulting text fragment.

Imperative and declarative approaches have their advantages and disadvantages. Therefore, hybrid approaches were developed to combine the advantages of both approaches. Hybrid transformation languages combine declarative and imperative language constructs. A developer must decide on the use of imperative or declarative constructs. Hybrid languages are, for instance, the ATLAS transformation language (ATL) [Jouault and Kurtev 2006; Jouault 2006] or the Epsilon transformation language (ETL) [Kolovos 2008]. They allow the definition of rules which consist of a header and a rule body. The rule header is declarative and describes the mapping between meta-model elements. The rule body allows the usage of commands to create models and calculates element values.

### 2.2.4 Epsilon Transformation Language

The Epsilon Transformation Language (ETL) [Kolovos 2008; Kolovos et al. 2015] is a powerful language for model transformations. This language is part of the Eclipse project Epsilon\(^1\). Epsilon provides languages and tools in order to realize typical tasks in the area of Model-Driven Engineering, such as code generation, model-to-model transformation, model validation, comparison, migration, and refactoring. The Epsilon framework supports EMF models but it is not limited to EMF at all because Epsilon defines its own type system enabling the connection to other technologies (e.g. XML or CSV). We use ETL in this work to describe different model transformations because ETL allows a concise and understandable description of transformations.

ETL follows a hybrid approach with declarative rules and imperative rule bodies. The structure of a rule is shown in Listing 2.1. A mapping rule begins with the keyword rule followed by a name. The head of a rule defines the mapping between meta-model elements. There is a left hand-side and a right hand-side.

\(^1\)http://www.eclipse.org/epsilon/
The left-hand side defines the domain of a mapping rule and is introduced by the keyword *transform* followed by a source parameter. The right-hand side defines the co-domain of a mapping rule and is expressed by the keyword *to* followed by one or more target parameters. The source and target parameter consist of a name and an EClass element of source and target meta-model, respectively. The defined source and target parameter can be used in a rule body. ETL supports rule inheritance that can be specified by the keyword *extends* followed by a rule name which reference another rule.

After the rule header, a rule body begins with an optional guard expression. The guard expression allows the definition of a condition which must be fulfilled (true) in order to execute a rule. After the guard expression, a rule uses imperative statements to describe additional mapping logic. Statements use the Epsilon Object Language (EOL). EOL is a model-oriented scripting language that combines the procedural style of JavaScript with model querying capabilities of OCL [OMG 2014b]. EOL uses a specific type system, which consists of primitive types (e.g. string, integer, real and boolean), collection types (e.g. bag, sequence, or set), native types, and meta-model elements (e.g. EClass from Ecore). Native types enable the embedding of Java types. Based on these types, there is a set of defined operations. Additional to the type system, EOL defines arithmetical, comparison and logical operators. Furthermore, there are statements for variable declarations, assignments and control flow (e.g. for, if, while).

2.2.5 Typical Usage of Transformations

The application field for model transformations is manifold. Generally, model transformations are used in the field of software development. Model transformations are a cornerstone in Model-Driven Software Development. Models can describe different software aspects and abstract from concrete source code for certain purposes. Application logic can be described in a domain-specific language, thus, domain experts can be involved in the development process. Furthermore, the formalization of models allows an easy correction of faulty design. Model transformations close the abstraction gap between domain models and executable source code.
A further application field for model transformations is the integration of software systems. Often, application systems cover only a specific task in a business process or development process. Systems must be integrated in order to support the whole business or development process. Software systems have often different data structures and functions because of their specialization. The integration of systems must overcome this heterogeneity. Transformations can be one mechanism to solve this heterogeneity problem. For instance, a transformation can be implemented as an interface allowing the import or export of data. Furthermore, integration platforms can use transformations for implementing software adapters.

A special application domain for transformations is the field of data integration. Data integration combines different data sources and provides a unified view on the data. The ETL process is an important task in this field. ETL refers to databases and stands for extract, transform, and load of data. The first part of an ETL process is the extraction of data from different source systems. In many cases, this is the most important part of ETL because extracting data correctly influences the success of subsequent processes. In the transformation phase, rules are applied to the data for different purposes. The load phase stores the data in a target system (e.g. data warehouse).

2.3 Summary

In this section, we introduced a set of basic concepts. Generally, we address model interoperability between meta-modeling environments. Thus, we firstly explained different concepts in the area of modeling. We focus on graphical models and modeling languages which are created in meta-modeling environments. These environments support a meta-modeling approach with a classical three-layer architecture.

The second part of this chapter referred to the concept of model transformation. The addressed M3B approach uses the concept of transformation to achieve interoperability between meta-model environments. Hence, we explained the concept of model transformation and surrounding aspects required for the M3B approach. The concept of model interoperability was not part of this chapter and will be introduced in the next chapter.
3 Interoperability between Meta-Modeling Environments

In this chapter, we introduce a general definition of model interoperability and conduct a study about interoperability between meta-modeling tools. The study shows the current degree of interoperability and research current approaches to achieve interoperability.

3.1 Definition of Model Interoperability

Ever since there are software systems interoperability is in research and in practice a subject of discussion. The word interoperability consists of two parts: “inter-operate” and “ability”. Inter-operate means that two systems can work together [Chen 2005] and the suffix signifies “the ability of a system [...] to work with or use the parts [...] of another system” [Gove 1993]. One basis for interoperability is the capability to exchange information between two or more systems and to use the information that has been exchanged [IEEE 1990]. The exchange of information between systems can be realized with various solutions.

Interoperability is the basis for integrating systems. Integration is also a common term in software and system development. Integration can be defined as the combination and coordination of separate things, elements or units into a whole so that they work together effectively [Gove 1993; Marwick et al. 2014]. Regarding the concept of interoperability, integrated systems must be interoperable in any form, but interoperable systems do not need to be integrated. Interoperability extends the border of already existing systems and enables the connection to other systems. Interoperability is often associated with loosely-coupled systems, where systems keep their autonomy [Molina et al. 2007]. In contrast to this, integration is characterized by a closely-coupled systems, where system are interdependent and difficult to separate from each other.

A further term in this context is migration. Generally, migration denotes a process of spatial movement. In information technology there are different application areas for migration, such as software systems, databases, application systems or hardware. A migration in the area of software systems is, for instance, updating from one major software release to the next highest software version. Already existing data, settings or specific extensions have to be transferred from the old software to the new software system.

In the context of this thesis, model interoperability deals with the exchange of models between meta-modeling tools. The exchange is realized as a migration of models and meta-models from one tool to another. The migration should be
an isomorphic relation in order to preserve the structure and semantics of models. However, the terms model interoperability (short interoperability), integration, model exchange and migration can be used as synonyms in this work.

3.2 Structure of the Study

3.2.1 Objective of the Study

The objective of this study is to investigate the state of the art in model interoperability. We want to investigate the degree of interoperability and we want to recognize approaches for the exchange of models [Kern 2014]. The study is focused on meta-modeling tools and approaches being used in the real world. The objective can be underpinned with the following two research questions.

**What is the degree of interoperability?** The first research question concerns the degree of interoperability. We want to analyze between how many of the involved tools an exchange of models is possible. Based on our experience, we assume that these possibilities are insufficient. The study should confirm this assumption.

**What approaches are used for interoperability?** The second question concerns the used model exchange approaches. There are already a variety of approaches in a theoretical context. However, we want to identify approaches which are used in practice. These approaches can help to identify possible requirements and solutions for the development of a suitable transformation approach.

In the subsequent section, we define the scope of this study. Afterwards, we describe the selection of appropriate meta-modeling tools and the analysis of these tools. We conclude with the presentation of the results and discuss their validity.

3.2.2 Scope of the Study

There are a variety of problems and solutions concerning tool interoperability. For instance, the annotated bibliography from Wicks [2006] shows various papers about interoperability between software development tools. In this section, we describe a set of dimensions which helps in structuring the study and narrowing down the scope of this study. The selection of these dimensions are based on a literature analysis and the knowledge of the author. The dimensions of the study are presented in Figure 3.1. These dimensions and their properties are not complete but sufficient for the study. Cells with a checkmark define the addressed investigation space. Gray cells are dimensions relating to the M3B approach and thus are interesting for a detailed analysis.

**Unification Mechanism**

One of the main reason for missing interoperability is the heterogeneity between artifacts. Heterogeneity between models can be, for instance, of syntactic or semantic
3.2 Structure of the Study

nature. To achieve model interoperability, it is necessary to overcome this heterogeneity and to find a unification between different structures. We can distinguish between the following two fundamental unification mechanisms.

**Common structure** One mechanism to realize interoperability is to avoid heterogeneity a priori by defining a common structure. The definition can be regarded as a development process for a standard. Such a standard defines, for instance, a common structure for models and meta-models, their semantics, and a specification for the exchange of models. If all systems conform to a standard, interoperability is guaranteed by this standard. Standards can address different aspects of exchange. For instance, in the modeling domain, there are standards which define a whole language (syntax, semantics and pragmatics). The Unified Modeling Language (UML) [Fowler 2003] is one example of this. Additionally, there are standards which define whole meta-modeling environments (e.g. Meta Object Facility (MOF) [OMG 2015a]) and corresponding exchange formats (e.g. XML Metadata Interchange (XMI) [OMG 2015c]). Meta-modeling tools which use MOF, UML and XMI as serialization syntax can exchange models and meta-models without problems (theoretically [Lundell et al. 2006]).
**Transformation** Another mechanism is the transformation of models and meta-models. A transformation defines a mapping between different structures to overcome heterogeneity. Similar to standards, transformations can address semantic or syntactic issues. If there is no standard available, a transformation is a powerful approach to exchange models. The mechanism of a common structure and a transformation are not mutually exclusive. A meta-modeling environment can implement a standard by using transformations in order to create models and meta-models conforming to this standard, but this is only possible up to a certain degree.

This study investigates possible common structures (standards) and transformation approaches.

**Model Level**

A meta-modeling tool has a modeling (M1) and a language level (M2). On the language level, a language engineer can define different modeling languages by using meta-models. These modeling languages can be used by a modeler at the modeling level to create models. Based on these two levels, we differentiate between the following two cases.

**Model exchange** This approach allows only the exchange of models. The exchange of languages is excluded in this approach. Generally there are two variants. The first variant supports the exchange of models as a generic structure independent of a specific modeling language. The generic exchange approach cannot ensure the interpretation of elements in the target tool. The second variant allows the exchange of models conforming to a specific modeling language. To realize the second variant, a common or standardized modeling language must be implemented in both meta-modeling tools. The exchange approach is then implemented against this specific language.

**Language and model exchange** This approach allows the exchange of models and languages. There are also two variants. The first variant supports a generic exchange of models and language elements independent of a meta-modeling language. The generic exchange approach cannot ensure the interpretation of elements in the target tool. The second variant allows the import of languages into a target tool. The imported languages are expressed in the meta-modeling language of the target tool. An important aspect of this variant is the language preservation between the participating meta-modeling environments. Language preservation relates to the meta-model and concrete syntax. Generally, models and languages from the source and target should be equal. This variant is suitable for meta-modeling tools.

The study investigates the exchange on both levels: model as well as meta-model level.
3.2 Structure of the Study

Topology

Meta-modeling tools exchange their data by using different topologies. The topology concept is originally from computer networks and is also used in software and system integration (e.g., Enterprise Application Integration [Gold-Bernstein and Ruh 2004]). The topology concept can be transferred to the area of meta-modeling tool integration. We can distinguish between the following topologies.

**Point-to-point** A simple topology is a point-to-point connection between two tools. In most cases, this is sufficient for the exchange between two tools. An example of a point-to-point connection is the export and import of models via a file between two tools. A connection is a direct point-to-point connection, if there is no recognizable transformation between source and target tool. Otherwise, we have an indirect point-to-point connection.

**Complex topologies** If more than two meta-modeling tools are involved in the integration solution, a point-to-point connection may be insufficient. For that reason, there are complex integration topologies such as star or bus topologies. A star topology often offers one common exchange format or interface to exchange models between all participating tools. The realization of a star topology requires an additional integration component, which is in the center of the integration architecture. This component controls the integration process and serves as a common structure for models. Another integration topology is a bus structure which can either use a common exchange structure or an individual defined exchange structure between each pair of tools. Realizations of a complex topology are, for instance, Model Bus [Hein et al. 2009] or BPM-X-Change [Kammermeier et al. 2011].

We study interoperability approaches which realize a direct point-to-point connection between tools and we omit the investigation of complex topologies.

Integration Layer

Integration of software requires the exchange of artifacts. Generally, there are different layers to access these artifacts. Integration layers often correspond to these access layers. Analogously to the integration of software development tools in Wasserman [1990], we can differentiate between the following typical layers.

**Data** Models and meta-models can be represented as data in files or databases. Hence, the integration can be realized on the data layer. Many tools enable file export or import. In this case, no complex integration architecture is necessary. The disadvantage of this approach is that the data serialization of models and meta-models is often complex. Hence, the processing of data implies a re-implementation of complex operations.

**Function** Above the data layer, many tools provide an API with different functions. The usage of the function layer is easier than the direct operation on
data layer because complex operations are encapsulated in functions. Typical functions are, for instance, the selection and creation of model elements. There are integration approaches using the function layer instead of the data layer.

**Presentation** The third layer is the presentation layer. The exchange on presentation layer often considers only the graphical representation of models. An example for this approach is the export as an image or the usage of Object Linking and Embedding (OLE). In this case, objects are represented in the target tool as a link to the source tool.

The study investigates the exchange on data layer and particularly the exchange via files.

### 3.2.3 Tool Selection and Analysis

**Tool Selection**

The selection of tools is an important aspect of this study because the selection has a strong influence on the study results. The selection is focused on tools using the term *meta-modeling tool*. The problem is that this term is more used in a theoretical context. Tool vendors often use other names for their tools. Hence, the search includes synonyms such as *meta-case tool*, *language workbench* and *diagram tool*. We also include the general word *modeling tool* in the search because most meta-modeling tools are often associated with this (simple) term. Based on the search results, we filter the initial set of tools by using the following criteria.

**Maturity level** The tool must fulfill a certain level of maturity. The tool must be installable and usable. This is necessary for the later analysis. Most tools are available as desktop applications but there are also tools implemented as web applications. Documentation is helpful but not obligatory.

**Graphical modeling** A further requirement concerns the concrete syntax of models. We select meta-modeling tools supporting the definition of graphical modeling languages. We exclude tools with textual languages.

**Heavyweight meta-modeling** The third criterion refers to the meta-modeling approach. A tool must support a heavyweight meta-modeling approach with a three-level model hierarchy.

**Modeling domain** The last criterion concerns the modeling domain. We select tools which have no specific domain, that is, they have a universal/generic character. Additionally, some tools can address the following modeling domains: software development, business process modeling and data modeling.

The initial search includes over 100 modeling tools. Out of this set, 63 tools are installable and 20 tools fulfill the previously defined criteria. The initial tool set with 63 tools is presented in Table A.1 in Appendix A. Finally, Table 3.1 shows the set of tools included in this study.
3.2 Structure of the Study

<table>
<thead>
<tr>
<th>Tool Name</th>
<th>Tool Vendor</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agilian</td>
<td>Visual Paradigm</td>
<td>4</td>
</tr>
<tr>
<td>ARIS Business Architect</td>
<td>Software AG</td>
<td>7.1</td>
</tr>
<tr>
<td>AToM(^3)</td>
<td>McGill University</td>
<td>2008</td>
</tr>
<tr>
<td>ConceptDraw</td>
<td>CS Odessa</td>
<td>9</td>
</tr>
<tr>
<td>Cubetto Toolset</td>
<td>Senture</td>
<td>1.7.1</td>
</tr>
<tr>
<td>Dia</td>
<td></td>
<td>0.97.2</td>
</tr>
<tr>
<td>Edraw Max</td>
<td>EdrawSoft</td>
<td>6.3</td>
</tr>
<tr>
<td>Enterprise Architect</td>
<td>Sparx Systems</td>
<td>9.3</td>
</tr>
<tr>
<td>Generic Modeling Environment</td>
<td>Vanderbilt University</td>
<td>10.8</td>
</tr>
<tr>
<td>iGrafix Process</td>
<td>iGrafix</td>
<td>2011</td>
</tr>
<tr>
<td>Lucidchart</td>
<td>Lucid Software</td>
<td>–</td>
</tr>
<tr>
<td>Maram Meta-Tools</td>
<td>University of Auckland</td>
<td>–</td>
</tr>
<tr>
<td>MetaEdit+</td>
<td>MetaCase</td>
<td>4.5</td>
</tr>
<tr>
<td>Microsoft Visio</td>
<td>Microsoft</td>
<td>2010 (14)</td>
</tr>
<tr>
<td>PowerDesigner</td>
<td>Sybase</td>
<td>16.1</td>
</tr>
<tr>
<td>ViFlow</td>
<td>ViCon</td>
<td></td>
</tr>
<tr>
<td>Visual Paradigm for UML</td>
<td>Visual Paradigm</td>
<td>9</td>
</tr>
<tr>
<td>Visualization and Modeling SDK</td>
<td>Microsoft</td>
<td>VS2012</td>
</tr>
<tr>
<td>yED</td>
<td>yWorks</td>
<td>3.9.2</td>
</tr>
</tbody>
</table>

Table 3.1: List of meta-modeling tools

**Tool Analysis**

The analysis starts with the installation of all tools. The focus of the investigation lies on the import and export functionality of each tool. Typically, a tool offers three interface layers which we can use for the investigation. These interfaces are: user interface, programming and data interfaces. We begin with the analysis of the user interface, particular tool menus, because tools often provide a menu entry for the import and export of modeling artifacts. Some tools offers an extra menu entry for the import and export, other tools offer this functionality as part of the load/open and save menu.

In addition to the user interface, we use available documentation to find exchange possibilities. The documentation can be a tool description which present possible features. Many tool vendors emphasize their import and export capabilities because this is an important feature for potential customers. Furthermore, tool manuals are helpful to figure out the exchange capabilities.

Some tools provide import and export capabilities via their programming interface and yet others provide a generator component which can be used to implement export and import scripts. These capabilities are excluded in the study with the exception if the export or import capability can be used for the import and export
without spending any additional amount of work (e.g. programming of generators or additional functions).

Furthermore, we restrict the investigation of the possibilities to exchange models. That is, we check only the exchange approach to understand the underlying principle but we do not investigate the quality or success of the approach. The investigation of quality would be too labor-intensive. We note every exchange capability and insert the name of the import and export possibility in Table A.2 in Appendix A. Based on this table, we present our findings of this study in the next section.

3.3 Results of the Study

3.3.1 Unification Mechanism

Common Structure

Many tools use the approach of a common structure to exchange their models. Each investigated tool supports a tool-specific data format which allows saving and loading of models within the same tool. Additional to the save and load format, some tools support a format which allows the exchange between different tools. Some tools use the Visio format which allows the exchange of models and meta-models between different tools. Additionally, there are standards allowing only the exchange of models conforming to a defined (standard) language. For instance, some tools enable the export and import of BPMN-XML. But these standards do not allow the exchange of languages. Finally, there is no common format – with the exception of Visio – that allows the exchange of meta-models and models between different tools.

Transformation

The exploration of transformation approaches is difficult because most tools implement their import and export functions as a black box component. Hence, we can only investigate transformations which are visible during the exchange process. We found the following two transformation approaches.

- Agilian, Visual Paradigm for UML and Business Process Visual Architect: These tools allow an export to Visio or an import of Visio models. The approach uses a transformation with wizard support which allows the configuration of mappings between elements of the modeling tool and Visio elements. The mapping is restricted to certain modeling languages.

- ARIS: This tool supports an import of Visio models. The import reads an XML configuration which describes a mapping between ARIS and Visio elements.

Both transformation approaches are not comparable with a powerful transformation approach such as Eclipse Epsilon Transformation Language (ETL) or Atlas Transformation Language (ATL).
3.3 Results of the Study

3.3.2 Model Level

Model Exchange

All tools support a generic exchange of models because each tool can save and load models as a tool-specific format. But this mechanism works only within the same tool (as save and load function). The exchange between different tools is not possible because the interpretation of the generic format is unclear in the target tool.

Some tools include pre-defined modeling languages. These languages are often standard languages in a certain domain such as BPMN, UML or EPC in the domain of business process modeling. Based on these pre-defined languages, some tools offer a language-specific exchange. An example is Agilian that allows the export of BPMN-XML and Business Process Visual Architect that allows the import of BPMN-XML. Regarding the unification mechanism, this approach follows the strategy of a common structure. The limitation to a pre-defined language is for meta-modeling tools unsatisfying. The following tools support a language-specific exchange:

- Agilian, Visual Paradigm for UML, Business Process Visual Architect: These tools allow the import of models conforming to pre-defined languages. Additionally, these tools allow the import of Visio documents with the problem that the Visio language is not really imported into these tools. The imported language elements are only represented in a separated icon library.

- ARIS: This tool allows the import of Visio models without the import of stencils, but ARIS offers a function which defines a mapping between Visio language elements and certain ARIS language elements.

- Edraw Max: This tool enables the import of Visio models. It is impossible to import Visio language elements.

- Lucidchart: This tool allows the import and export of Visio models. The export was not testable in the free version. The import transforms only graphical elements on model level without corresponding Visio language elements.

- Dia: Dia allows the import and export of Visio models without Visio language elements.

Language and Model Exchange

Each tool offers a generic data structure to represent models and language definitions. For instance, MetaEdit+ or Dia use a generic format to save models and languages as graphs. The support of a generic exchange approach is not surprising because each tool uses this mechanism for saving and loading their models and languages as files. Analogous to the generic model exchange, this mechanism works
only within the same tool. The exchange between different tools is not possible because the interpretation of the generic format is unclear in the target tool.

Additional to the generic approach, some tools support the exchange of models as well as languages between different tools. The target tool knows how to interpret the models and languages of a source tool. Some tools support the import of a whole language. Other tools can import only a subset of language elements which are used in models. But this approach leads to the problem that the import only considers a part of a language. The following tools support this approach.

- **ConceptDraw**: ConceptDraw enables the import of Visio models. Additionally to the import of models, ConceptDraw can import language elements (Visio masters). ConceptDraw indirectly imports master elements via the model import. ConceptDraw allows also the export of models to Visio but without language elements.

- **iGrafix**: This tool allows the import of Visio models by using the clipboard. The tool can also import stencil elements being used in the imported model.

### 3.3.3 Degree of Interoperability

An important question of this study concerns the degree of interoperability. For this purpose, we investigate the exchange capabilities of all participating tools and derive possible connections between all tools. Table 3.2 shows the export and import connections of each tool in a matrix. The vertical axis is the source tool and the horizontal axis is the target tool. Generally, the source tool exports models and the target tool imports models. For instance, there is a directed connection from Visio to ConceptDraw. That is, Visio can export models and ConceptDraw can import these Visio models. We differentiate the connections in dependency of their model level. A rectangle $\square$ is a connection that supports the exchange of models as well as languages. The plus sign $+$ stands for an approach which supports only the exchange of models. We combine the rectangle and plus sign $\square+$, if a tool supports a language-specific model exchange as well as an exchange of models and languages.

The diagonal in the matrix shows that each tool allows the exchange of their languages and models with itself because each tool can save and load its own language definitions and models. Additionally to this, many tools allow a language-specific model exchange. There are only three connections allowing the exchange of languages and models between different tools. The matrix also shows that Visio plays a key role because many tools allow the import of Visio models.

Overall, there are $20 \times 20 = 400$ possible directed connections in this matrix. We assume that the export and import within the same tool is a basic feature to save and load models and languages. For this reason, we exclude the diagonal in our calculation. Thus, we have $400 - 20 = 380$ possible connections. Out of these 380 connections, there are 30 connections between different tools. This makes a ratio of $31/380 = 8.2\%$. There are 27 connections that allow a language-specific model exchange. This is a ratio of $28/380 = 7.4\%$. Regarding the exchange of models and
3.3 Results of the Study

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Agilian</td>
<td>⊕</td>
<td></td>
<td>⊕</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARIS BA</td>
<td></td>
<td>⊕</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AToM³</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Business Process VA</td>
<td></td>
<td></td>
<td></td>
<td>⊕</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ConceptDraw</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cubetto Toolset</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dia</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Edraw Max</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enterprise Architect</td>
<td>+</td>
<td>+</td>
<td></td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GME</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iGrafix Process</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lucidchart</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maram Meta-Tools</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MetaEdit+</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Microsoft Visio</td>
<td>+</td>
<td>+</td>
<td></td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PowerDesigner</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ViFlow</td>
<td>+</td>
<td>+</td>
<td></td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VP for UML</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VMSDK</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>yED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.2: Model exchange between meta-modeling tools

languages, there are only three connections. This is a ratio of $3/380 = 0.8\%$. In summary, the degree of interoperability between the investigated modeling tools is quite low.

3.3.4 Further Observations

Generally, we identified different data formats for realizing the exchange of models. One known exchange format for models is XML Metadata Interchange (XMI) [OMG 2015c]. We consider XMI in our study but we are not focused on this format because of the close relationship to MOF, EMF and UML. Other meta-modeling tools outside the OMG scope do not use XMI for realizing the exchange of their models. Another mechanism to exchange models is the usage of graph formats, such as Graph Exchange Language (GXL) [Winter et al. 2002] or GraphML [Brandes et al. 2002]. Graph formats are suitable for the exchange because models and
meta-models can be regarded as graphs. For instance, MetaEdit+ uses an adapted version of GXL for serializing their models and meta-models but no other tool in the study can import this graph format. yEd can import GraphML but no investigated tool supports the export. A further observation is that some tools allow the export as Excel data. This is not for the exchange between modeling tools, but rather than a format to support the creation of reports. Besides the possibilities to exchange meta-models and models, there are a lot of language-specific formats, depending on the tool domain. For instance, many tools in this study support typical formats in the business modeling domain, such as BPMN-XML, BPEL [OASIS 2007] or XPDL [WFMC 2012].

Another observation is that more tools allow the import than the export of models. There could be a strategic reason for this. Tools support the import because vendors want to increase the usage of their tools and often it is necessary to import data in order to replace other tools. The export is undesirable because tool vendors try to bind their customers to a certain tool. Nevertheless, the difference between import and export possibilities are marginal, which leads to the conclusion that missing interoperability is not only a strategic but also a technological problem.

The last observation concerns the transformation capabilities. Some tools allow the definition of mappings between models or languages in a simple and limited way. But these tools do not provide powerful transformation approaches.

### 3.3.5 Threats to Validity

The degree of interoperability between the investigated tools lies between 0.8% and 8.2%. If we take a look at our study restrictions, interoperability could be higher than the values measured. We focused on a limited set of tools in selected domains. Maybe other tools in other domains provide a higher interoperability. Furthermore, we only looked for interoperability mechanisms provided by the tool itself. Some meta-modeling tools provide a generator framework which allows the generation of any exchange format. Furthermore, we excluded external tools/components (e.g. BPM-X-Converter) which also allow the migration of models.

In contrast to this, we can argue for a lower value of interoperability. We only investigated the opportunity to import and export models. We cannot say anything about the quality of a mechanism. We assume that the export and import of data can be faulty. Furthermore, some tools relate very close to Visio. Hence, these tools can implement the import and export to Visio easily. This is maybe similar to MOF-implemented meta-modeling tools. If we would not consider the exchange with Visio, the interoperability would go against zero.

### 3.3.6 Conclusion

In this section, we presented a study about interoperability between meta-modeling tools. The study included 20 tools in the area of software development, business process and data modeling. In the first part of this study, we defined the investigation scope. In the second part, we analyzed the tools and presented the results.
Regarding the first research question about the degree of interoperability, we can give the following answer. Depending on the considered approach, the degree of interoperability lies between 0.8% and 8.2%. This low degree of interoperability provides a good motivation for the topic of this work.

Regarding the second question about the approaches used, we can give the following answer. Many tools support a language-specific model exchange and use a common structure (standard format) which is specific for a (standardized) language. Furthermore, many tools allow the generic serialization and deserialization of models and languages as generic graph formats. Only a small number of tools use a transformation approach. In this case, models and languages are transformed in a very limited way. We focus on this issue and present a transformation-based approach which supports the exchange of models as well as meta-models.

3.4 Interoperability in other Technologies

Interoperability is a general issue in many other technological domains. Different technologies have different purposes and strengths in certain application areas. Often, the combination of technologies is convenient to achieve an integrated solution matching an addressed problem. For this reason, there are approaches enabling interoperability between different technologies. We investigated different interoperability approaches and recognized a common solution pattern to integrate different technologies. The idea is to adapt and apply this common solution pattern to the field of meta-modeling tools.

3.4.1 Technologies with a Three-Level Architecture

Analogous to the modeling space, other technologies also use a hierarchy to structure technology-specific entities. Such hierarchies have a high similarity to a model hierarchy. Similar to a model hierarchy, these hierarchies realize a three-level architecture [Kurtev et al. 2002; Kern and Kühne 2007b]. The highest layer forms the core of a technology. On this layer, a technology implements generic capabilities such as the storage of entities and the structure description of entities including their semantics. The second layer of a hierarchy allows the description of a specific problem and the usage of a technology. Developers can express a corresponding entity structure and can build a solution for their specific problem. The bottom layer is responsible for the runtime and holds concrete entities. The layer concept enables the reuse of generic capabilities and problem-specific solutions. Figure 3.2 shows on the left hand side the model hierarchy and on the right hand side different examples of technologies implementing a three-level architecture.

Relational databases The main purpose of this technology is the management and storage of relational data. A relational database consists mainly of tables. Data is located on the first layer and represents an instance of a previously defined database schema. A database schema describes constraints on data and defines valid data sets. A schema is located on the second layer and is
expressed by a data definition language. This definition language is specified on the third layer [Date 2008].

**Extensible Markup Language** Similar to relational databases, XML also allows the storage of data but in contrast to relational databases, XML supports the storage of semi-structured data. Furthermore, XML is more suitable for the exchange and transport of data over the Internet. Generally, a XML document consists of XML tags and values. The general structure of well-formed XML documents is defined in the XML data/information model [W3C 2008]. Additional to the well-formed character, a document can be valid or conform to a specific schema. A schema is expressed by an XML schema language and describes constraints for valid documents. XML data is located on the first layer, an XML schema is defined on the second layer and the corresponding schema definition language is specified on the third layer.

**Object-oriented languages** Object-oriented programming languages (e.g. Java or C#) allow the definition of classes describing the structure and behavior of objects. In contrast to databases and XML, the main purpose of a programming language is the processing of data, execution of commands, and implementation of algorithms. Objects are located on the first layer and are instances of classes. Classes are located on the second layer and are expressed with a programming language. The definition of a programming language is located on the third layer.

<table>
<thead>
<tr>
<th>Model space</th>
<th>Relational databases</th>
<th>Object-oriented programming</th>
<th>XML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meta-metamodel</td>
<td>Data definition language</td>
<td>OO language</td>
<td>XML schema language</td>
</tr>
<tr>
<td>conform to</td>
<td>conform to</td>
<td>instance of</td>
<td>valid</td>
</tr>
<tr>
<td>Meta-model</td>
<td>Database schema</td>
<td>Classes</td>
<td>XML schema</td>
</tr>
<tr>
<td>conform to</td>
<td>instance of</td>
<td>instance of</td>
<td>valid</td>
</tr>
<tr>
<td>Model</td>
<td>Data</td>
<td>Objects</td>
<td>XML document</td>
</tr>
</tbody>
</table>

Figure 3.2: Technologies with a three-level architecture

### 3.4.2 Interoperability Approaches

Based on the previously presented technologies, we describe three different interoperability solutions. All solutions support the exchange of data between equivalent
data hierarchies. The exchange approach uses the same transformation pattern. Each solution transforms artifacts from a source to a target on the second layer. Depending on this transformation, there is a transformation on the first layer which migrates artifacts between source and target.

**Java Architecture for XML Binding**

Java Architecture for XML Binding (JAXB) allows the combination of XML and Java and provides a Java programming interface for XML data [McLaughlin and Edelson 2006]. Generally, the framework enables the serialization of Java objects as XML documents and vice versa. The JAXB framework realizes a transformation which transfers Java classes to an XML schema and Java objects to XML documents as well as the reverse direction. Figure 3.3 shows this transformation approach. A binding on the second layer describes correspondences between XML schema and Java classes. Based on this binding, JAXB can serialize Java objects as XML data or can load XML documents as Java objects. The serialization and deserialization are implemented on the first layer by a framework component during the run-time of a Java program. JAXB denotes the process of serialization as marshalling and unmarshalling, respectively.

![Figure 3.3: Transformation approach in JAXB](image)

**Hibernate Framework**

Hibernate enables the combination of Java and relational databases [Bauer et al. 2015]. The framework creates Java objects from relational data and vice versa. The created objects are instances of classes which correspond to a database schema. The corresponding concept of this framework is an object-relational mapping (ORM or OR mapping) [Barry and Stanienda 1998]. The Hibernate approach is similar to JAXB. Hibernate creates Java classes from a relational database schema. The opposite way is also possible, i.e. based on annotated Java classes, Hibernate can generate a database schema. At run-time the Jawin component is responsible for data exchange between the relational database and Java objects. The data exchange depends on the concrete mapping between database schema and Java classes.
Jawin – Java/Win32 Interoperability

Jawin is a software bridge which facilitates the connection between Java and Microsoft COM or Win32 DLLs [Jawin 2005]. The exchange of data between Java and COM components uses the same transformation approach as JAXB and Hibernate. Firstly, Jawin reads the interface description of a COM component or DLL and generates equivalent Java classes. The mapping between the generated Java classes and methods in the COM components is described as annotations in Java classes. At runtime, a bridging component interprets the Java annotation and translates Java objects into COM objects and vice versa. The component passes Java statements to the COM/DLL application. Furthermore, the component is responsible for the object life-cycle in COM components.

3.5 Summary

In this chapter, we investigated the current state of the art in field of interoperability between meta-modeling tools. Firstly, we explained interoperability and related terms because the aspect of interoperability is manifold and has a broad meaning. We defined interoperability as the ability to exchange models and meta-models between meta-modeling tools. For this purpose, models and meta-models needs to be migrated between participating tools.

We investigated the degree of interoperability and analyzed typical approaches for realizing the exchange of models. Overall, the study investigated 20 meta-modeling tools. An important exchange approach is the usage of a common structure which is often implemented as a standardized data format. In general, we can conclude, that there is no common data structure for the exchange of models as well as languages, but we found that some tools use the Visio format for the exchange of models and languages. Besides the approach of a common structure, the study investigated possible transformation-based exchange mechanisms. Some tools offer a possibility to use a transformation-based approach but the exchange of languages were not adequately supported by these approaches. Furthermore, the transformation approaches used are not comparable with modern transformation solutions.

Regarding the degree of interoperability, we found that the exchange possibility is quite low. Less than 8% of all possible connections between all 20 tools allow an exchange of models without languages. Less than 1% enable the exchange of models and languages. We conclude that there is a lack of interoperability between tools. This lack provides the motivation of this work. The purpose of this thesis is the development of an approach or mechanism which offers the possibility to increase the interoperability in an efficient way.

In the last part of this chapter, we identified a solution approach which is used for the creation of interoperability in other technology domains. These technologies are also characterized by a three-layer architecture similar to a model hierarchy. The assumption is that this approach can be used to tackle the interoperability problem between meta-modeling tools.
4 M3-Level-Based Bridge

This chapter presents the central idea of this thesis. We describe the approach of M3-level-based bridges to solve the problem of model interoperability between meta-modeling tools. We provide a theoretical definition, describe different features, discuss various development aspects, and present a concept for the combination of different M3-level-based bridges.

4.1 Transformation Approach

4.1.1 Requirements for Model Interoperability

Firstly, we define a set of requirements for the exchange approach. This requirement description should improve the understanding of the addressed model interoperability problem as well as the suggested solution. In the context of this work, model interoperability is the possibility to migrate models from one meta-modeling tool into another meta-modeling tool. Thus, the first requirement is that a source tool can export models and a target tool can import these models. The exchanged models between source and target environment must be equal. That means, source and target models must include equal model elements with the condition that both models are expressed in the same modeling language. The language in both environments must be expressed with the meta-modeling language of the source and target environment, respectively. The equality refers mainly to the syntax of models. Furthermore, the equality between source and target model should indirectly preserve their semantics. Figure 4.1 shows an example of the required equality between models. The left (source) model is in MetaEdit+ and the right (target) model is in Visio. Both models are expressed in the EPC modeling language. The EPC modeling language is defined in the relevant meta-modeling language of MetaEdit+ or Visio. Both models contain the same elements with equal properties.

![Figure 4.1: Equality of source and target model](image)

(a) EPC in MetaEdit+  (b) EPC in Microsoft Visio
Another requirement is the generic application of the exchange approach. The approach should allow the exchange of every model expressed by arbitrary modeling languages. This requires that the exchange approach must consider models as well as the corresponding language definition. There are two possibilities on the language level. The first case is that a language definition is available in the source tool. This case requires the creation of an equivalent language definition in the target tool. The second case is that a language exists in the source and target tool. In that case, the creation of a language in the target tool is unnecessary but there must be a mapping between source and target language elements.

The next requirement concerns the underlying mechanism for the exchange approach. Generally, the exchange problem between different tools is caused by the heterogeneity of modeling languages. This heterogeneity stems from tool-specific meta-modeling languages and different meta-models implementing the same modeling language. Generally, there are two mechanisms to overcome this heterogeneity. The first mechanism is the usage of a common and standardized structure. If all tools implement this standard, the tools can interpret the exchanged models. Hence, the usage of such a common structure guarantees interchangeability. The second mechanism is the usage of a transformation which converts models between different tools. We want to use the transformation-based mechanism. The reasons for this decision are manifold. The first argument is derived from the study about interoperability in Section 3. We found that there is no common structure or standard which has been established between different tools. Based on this finding, the development of another standard would be questionable. The second argument stems from the principal creation and handling of heterogeneity. Standardization achieves interoperability by avoiding heterogeneity a priori. In this case, standardization requires a unification of meta-metamodels and meta-models. But this is difficult to realize because the exchange approach should address different meta-modeling tools with different meta-metamodels and different meta-models. Because heterogeneity is difficult to predict in advance, a transformation-based approach is more suitable to manage heterogeneity after its creation.

Further requirements relate to classical integration aspects. The first aspect is the integration layer. Meta-modeling tools are applications which are structured into the following three layers: data, function, and presentation. Based on these layers, there are different possibilities to integrate tools. In this work, we focus on the data and function layer because the most meta-modeling tools offer access via files or an API. Furthermore, the integration should be loosely-coupled and the exchange should be non-invasive without changing tools [Stefan et al. 2013]. And the last requirement concerns the integration topology. The exchange approach should use a point-to-point topology. The point-to-point topology is easy to implement and sufficient in many cases. If there is a meta-modeling tool suitable as a common (pivot) element, the point-to-point integration allows for extension to a star topology.
4.1 Transformation Approach

4.1.2 Transformation Description

The basic idea of an M3-level-based bridge is the conversion of models and metamodels from one model hierarchy into another model hierarchy. Figure 4.2 shows an overview of the M3B approach. An M3B consists of two transformations: an M2-transformation and an M1-transformation. The M2-transformation reads source meta-models and creates target meta-models. The M2-transformation is defined against both meta-metamodels. Additionally, the executed M2-transformation creates a transformation instance or trace which contains connections between the transformed meta-model elements. The M1-transformation reads source models and creates target models. The mapping of the M1-transformation depends on the transformation instance created by the previously executed M2-transformation. The dependency between both transformations is depicted as an overlapping of both gray rectangles in Figure 4.2.

The M2-transformation \( \tau_2 \) reads a source meta-model \( mm_s \in MM_s \) and produces a corresponding target meta-model \( mm_t \in MM_t \). The transformation comprises a set of transformation rules defining a mapping between a source and target meta-metamodel \( MMM_s \) and \( MMM_t \). The M2-transformation must implement a language-equivalent mapping. To fulfill this constraint, the transformation must define a mapping between equivalent meta-modeling concepts. A source metamodel \( mm_s \in MM_s \) is language-equivalent to a target metamodel \( mm_t \in MM_t \), if both meta-models \( mm_s \) and \( mm_t \) define the abstract syntax \( A \) of the same modeling language \( L \). The relationship between meta-model and abstract syntax is
defined as \( \sigma \) (see Equation 2.4). The language-equivalence between meta-models can be formally defined as the relation \( \sim_L \):

\[
\sim_L := \{(mm_s, mm_t) \in MM_s \times MM_t \mid 
\sigma(mm_s) = \sigma(mm_t) = A \text{ is part of } L\}. \tag{4.1}
\]

Application of the language-equivalence relation \( \sim_L \) enables a restriction of the M2-transformation. The M2-transformation is defined as a function \( \tau_2 \) between source and target meta-models. The target meta-model \( mm_t \), produced by the transformation function \( \tau_2(mm_s) \), must be language-equivalent \( \sim_L \) with the source meta-model \( mm_s \). Formally, we define this constraint on \( \tau_2 \) as follows:

\[
\tau_2 : MM_s \rightarrow MM_t, mm_s \mapsto mm_t
\]

\[
\tau_2(MM_s) := \{\tau_2(mm_s) \mid \tau_2(mm_s) \sim_L mm_s\}. \tag{4.2}
\]

The M2-transformation \( \tau_2 \) influences the M1-transformation \( \tau_1 \). The coupling of both transformations \( \tau_2 \) and \( \tau_1 \) requires a restriction allowing only the M1-transformation of models, if there is an M2-transformation between corresponding meta-models. We define the M1-transformation as a function with two parameters. The first argument is the source model \( m_s \) and the second argument is the M2-transformation \( \tau_2 \). Formally, we define the M1-transformation \( \tau_1 \) as follows:

\[
\tau_1 : M_s \times \tau_2 \rightarrow M_t, m_s \times \tau_2 \mapsto m_t.
\]

\[
\tau_1(M_s, \tau_2) := \{\tau_1 | (mm_s, mm_t) \in \tau_2\}. \tag{4.3}
\]

The dependency between M2- and M1-transformation is defined in the following equations. The M1-transformation \( \tau_1 \) requires a previously executed M2-transformation. That is, there must be an M2-transformation \( \tau_2 \) between a pair of source meta-model \( mm_s \in MM_s \) and target meta-model \( mm_t \in MM_t \). We define this constraint as follows:

\[
\tau_1(M_s, \tau_2) := \{\tau_1 | (mm_s, mm_t) \in \tau_2\}. \tag{4.4}
\]

Next, we substitute the source meta-model and target meta-model in this constraint because the arguments of the transformation \( \tau_1 \) do not include source and target meta-models. Firstly, the source meta-model \( mm_s \) is substituted by the \textit{conform to} function \( \chi \). Secondly, the target meta-model \( mm_s \) is substituted by the application of the M2-transformation \( \tau_2 \) with the source meta-model \( mm_s \) as input argument. Both substitutions are described in the following equations:

\[
\tau_1(M_s, \tau_2) := \{\tau_1 | (\chi(m_s), mm_t) \in \tau_2\}
\]

\[
\tau_1(M_s, \tau_2) := \{\tau_1 | (\chi(m_s), \tau_2(mm_s)) \in \tau_2\}. \tag{4.5}
\]

Last step is the substitution of the source meta-model \( mm_s \) in the M2-transformation \( \tau_2 \). The source meta-model is replaced by the \textit{conform to} function \( \chi \) which needs a source model \( m_s \) as input and returns the corresponding source meta-
4.1 Transformation Approach

The following equation describes the derived definition of the M1-transformation $\tau_1$.

$$\tau_1(M_s, \tau_2) := \{ \tau_1 | (\chi(m_s), \tau_2(\chi(m_s))) \in \tau_2 \}. \quad (4.7)$$

Finally, Equation 4.5 shows the formalization of the M2-transformation $\tau_2$ and Equation 4.7 shows the formal description of the M1-transformation $\tau_1$ in dependency of the previously defined M2-transformation.

4.1.3 Transformation Algorithm

In this section, we want to present the M3B approach from an imperative viewpoint to demonstrate the transformation process. For this purpose, we describe a simplified algorithm. The transformation algorithm can be divided into two parts.

The first part implements the M2-transformation. Algorithm 1 shows this transformation as pseudo code. Input is a source meta-model $mm_s$ and output is a target meta-model $mm_t$. The mapping of meta-model elements $\tau_{2i}$ (also denoted as transformation instance) is implemented as a map. This map forms the basis for the equivalent function which returns an element $E_t$ from the target meta-model corresponding to an element $E_s$ from the source meta-model.

The algorithm starts with $\text{METAMODEL\-TRANSFORMATION}(mm_s, mm_t)$ in line 4. Firstly, $mm_s$ is loaded, $mm_t$ is created and $\tau_{2i}$ is initialized. After that, this function implements a transformation rule which iterates over all object types ($ot_s$) in the source meta-model and executes $\text{TRANSFORM\-OBJECT\-TYPE}(ot_s)$. This function creates an object type $ot_t$ and adds this object type to the target meta-model $mm_t$. Finally, the connection between the source object type $ot_s$ and target object type $ot_t$ is added to the map $\tau_{2i}$. This map is the return value and serves later as input for the M1-transformation. Finally, the algorithm stores the created target meta-model.

The M1-transformation is presented in Algorithm 2. The entry point for this transformation is $\text{MODEL\-TRANSFORMATION}(mm_s, m_s, mm_t, m_t)$ in line 5. This function loads the source model $m_s$ and creates the target model $m_t$. Afterwards, the meta-model transformation is executed in line 8 and the transformation instance is stored in $\tau_{2i}$. Based on this map, the equivalent function returns a target meta-model element $e_t$ which corresponds to an element $e_s$ from the source meta-model. Next, a transformation rule is executed. Analogous to the M2-transformation, this rule iterates over all objects and calls the function $\text{TRANSFORM\-OBJECT}(o_s)$. This function queries the object type $ot_s$ of the object $o_s$ and requests the corresponding target object type $ot_t$. Next, the algorithm instantiates the object type $ot_t$ and creates the object $o_t$. Finally, the object $o_t$ is added to the target model $m_t$.

Note, the transformation rules in this algorithm depend on the participating model hierarchies. For the sake of simplicity, we use a trivial meta-metamodel consisting of one concept which is denoted as object type. Furthermore, we use the same source and target model hierarchy. Normally, the source and target model hierarchy are different as the purpose of an M3B is the integration of different meta-
modeling environments. Nevertheless, the principle of this algorithm is adaptable to more complex meta-modeling environments. The adaption mainly concerns the transformation rules in the algorithm. The *equivalence* function and the interdependence between both transformations is independent of concrete meta-model environments.

**Algorithm 1** Transformation of meta-models

1: \(mm_s\): source meta-model; \(mm_t\): target meta-model
2: \(\tau_{2i}\): map of meta-model elements with \(MM_s \times MM_t\)

3: function MetamodelTransformation\((mm_s, mm_t)\)
4: \(mm_s := \) load source meta-model
5: \(mm_t := \) create target meta-model
6: \(\tau_{2i} := \) initialize map
7: for all \(ot_s \in \) object types in \(mm_s\) do
8: \(\) TransformObjectType\((ot_s)\)
9: end for
10: \(\) return \(\tau_{2i}\)
11: end function

12: function TransformObjectType\((ot_s)\)
13: \(ot_t := \) create a new target object type
14: \(\) add \(ot_t\) to the target meta-model \(mm_t\)
15: \(\) add tuple \((ot_s, ot_t)\) to the map \(\tau_{2i}\)
16: end function

**Algorithm 2** Transformation of models

1: \(m_s\): source model; \(m_t\): target model
2: \(mm_s\): source meta-model; \(mm_t\): target meta-model
3: \(\tau_{2i}\): map of meta-model elements with \(MM_s \times MM_t\)
4: \(\text{equivalent}: MM_s \to MM_t\); \(\text{equivalent}(es): \) if \((es, et) \in \tau_{2i}\) then return \(et\)

5: function ModelTransformation\((mm_s, m_s, mm_t, m_t)\)
6: \(m_s := \) load source model
7: \(m_t := \) create target model
8: \(\tau_{2i} := \) MetamodelTransformation\((mm_s, mm_t)\)
9: for all \(os \in \) objects in \(m_s\) do
10: \(\) TransformObject\((os)\)
11: end for
12: end function

13: function TransformObject\((os)\)
14: \(ot_s := \) get object type of \(os\)
15: \(ot_t := \) equivalent\((ot_s)\)
16: \(ot_t := \) create instance of object type \(ot_t\)
17: \(\) add \(ot_t\) to the target model \(m_t\)
18: end function
4.1 Transformation Approach

4.1.4 Transformation Characteristics

The M3B approach can be characterized by a set of transformation properties presented in Section 2.2.2. This characterization should help to classify the M3B into existing model transformation approaches.

M2-Transformation

**Model-to-model and model-to-text transformation** The M2-transformation is a model-to-model transformation because both input and output are models. Regarding the model hierarchy, these models are meta-models and their meta-models are meta-metamodels.

**Homogeneous versus heterogeneous transformation** A heterogeneous transformation is between different meta-models, otherwise it is a homogeneous transformation. Regarding the model hierarchy, the meta-models of the M2-transformation are meta-metamodels in a hierarchy. The M2-transformation is between two different meta-models. Thus, the M2-transformation is a heterogeneous transformation.

**Endogenous versus exogenous transformation** An exogenous transformation is between two different meta-metamodels, otherwise it is an endogenous transformation. The decision about this property is difficult. Regarding the model hierarchy, there is no real meta-metamodel. But if there would be a meta-metamodel then we assume that both would be different. Hence, we can assume the M2-transformation is exogenous.

**Horizontal versus vertical transformation** A horizontal transformation preserves the abstraction level, otherwise the transformation is a vertical transformation. The M2-transformation is horizontal because both meta-models represent the same modeling language. There is no change of the abstraction level. The M2-transformation can be regarded as a migration of meta-models between different model hierarchies. A migration is a typical example for a horizontal transformation [Visser 2001].

**Unidirectional versus bidirectional transformation** The M2-transformation can be bidirectional but we regard the M2-transformation as a unidirectional transformation from a source to a target hierarchy.

**Higher-order transformation** The M2-transformation is a simple model transformation because there is no dependency to another model transformation.

M1-Transformation

**Model-to-model and model-to-text transformation** The M1-transformation is also a model-to-model transformation because both input and output are models with corresponding meta-models.
Homogeneous versus heterogeneous transformation  The transformation is heterogeneous because the participating meta-models are not identical. Both meta-models are expressed by different meta-modeling languages. Nevertheless, the M2-transformation produces meta-models which are language-equivalent. That is, both meta-models represent the same abstract syntax of a modeling language. If we take the language-equivalent relationship between meta-models instead of the identical relationship, the transformation is homogenous.

Endogenous versus exogenous transformation  The M1-transformation is exogenous because the participating meta-metamodels are different.

Horizontal versus vertical transformation  Analogous to the M2-transformation, the M1-transformation is a horizontal transformation because the input and output models are on the same abstraction level.

Unidirectional versus bidirectional transformation  Analogous to the M2-transformation, the M1-transformation is defined in this context as a unidirectional transformation from a source to a target hierarchy.

Higher-order transformation  The M1-transformation is a higher-order transformation because the M1-transformation takes as input the M2-transformation. The assumption is that the M1-transformation is a transformation synthesis because the M1-transformation reads the transformation instance of the M2-transformation. Equation 4.7 confirms this statement.

4.2 Features of M3-Level-Based Bridges

4.2.1 Direction

A basic feature of M3Bs is the transformation direction. We assume that we have unidirectional transformations. The direction of a transformation is defined from the source to the target model. The direction feature of an M3B must consider both transformation levels. We identified three different variants for the direction. The first variant is illustrated in Figure 4.3(a). This variant allows only the export of meta-models and models from source to target.

In addition to the first variant, the second variant allows the re-import of models from the target back to the source. Figure 4.3(b) illustrates this variant. The condition for re-import is that the target model must conform to a target meta-model which was previously exported by an M2-transformation.

The third variant allows the transformation of meta-models and models in both directions. This variant is illustrated in Figure 4.3(c). The transformation of models is without the restriction from the second variant. This variant requires four transformations, two from source to target and two in the reverse direction. Note, if the transformation systems support bidirectional transformations, this variant needs only two transformations on each level.
4.2 Features of M3-Level-Based Bridges

4.2.2 Configuration

The M2-transformation implements the transformation of meta-models. Regarding the configuration of M3Bs, there are two cases. In the first case, the M2-transformation implements a fixed set of rules. Figure 4.4(a) shows this case. There is no way to influence the transformation between the meta-models because the rule set is not changeable.

The second case allows the adaption of transformation rules at M2-level. Figure 4.4(b) shows this case. The bridge offers a default configuration defining a default mapping. This default mapping can be adapted by an additional description of mapping rules. The adaption influences the meta-model transformation up to a certain degree. For instance, if there are alternative ways to map meta-model elements, the configuration allows the selection of a certain mapping variant. The advantage of a configuration is that the M2-transformation can be easily adapted for a certain purpose. The disadvantage is that configuration feature increases the development complexity of a bridge because the bridge must consider different transformation possibilities.

Figure 4.4: Configuration of meta-model transformations
4.2.3 Output

A bridge supports two output variants. The first variant is illustrated in Figure 4.5(a). In this case, the bridge is used to create a target meta-model. Additional to this meta-model, the M2-transformation creates a transformation instance which stores the mapping between participating meta-models elements. After this, the M1-transformation reads the transformation instance, loads the source model and creates a target model.

In the second variant, the meta-model transformation does not create a target meta-model. Figure 4.5(b) shows this case. The M2-transformation provides only a transformation instance between meta-model elements in order to control the M1-transformation later. The M1-transformation reads the transformation instance and creates a target model. This variant is typically for the model migration between fix meta-models because the repeating transformation of meta-models is not necessary.

![Diagram](a) Creation of meta-models and models (b) Creation of models

Figure 4.5: Transformation output

4.2.4 Validation

The validation feature checks constraints in a target model. Figure 4.6 illustrated the validation feature. In the case that a target model should be re-imported into the source model hierarchy (see Section 4.2.1), a validation of the target model

![Diagram](Validation support)
against the source meta-model can be helpful. The reason for this validation lies in the meta-model transformation. In some cases, there can be a loss of information during the M2-transformation. This transformation loss allows the creation of target models conforming to the target meta-model but the transformation of this target model back into the source (re-import) can fail.

The validation of target models against the source meta-model works as follows. During the creation of a target model, a validation component transforms the target model into a source model. This source model is validated against the source meta-model. Possible errors relate to the source model and must be translated to the target model. The component can present these errors and can suggest improvements.

4.3 Development of M3-Level-Based Bridges

4.3.1 Transformation System

The execution of an M3B requires a transformation system. Based on the analysis of M3Bs, we identified a set of basic system components. Figure 4.7 shows an overview of the transformation system. Interfaces to meta-modeling environments are one part of the system. An interface serves as a data provider for the transformation components and is responsible for the import and export of models and meta-models. Depending on the transformation direction, interfaces must support reading and writing of models.

One further part of the system is the transformation component. This component is responsible for the execution of the M2- and M1-transformation. The number of transformations depends on the supported directions of a bridge (see Section 4.2.1).

Figure 4.7: Abstract M3B transformation system
The coupling of both transformations is realized by a mapping repository. The repository stores transformation instances of the M2-transformation. A transformation instance comprises relations between participating source and target meta-models. The M1-transformation uses a transformation instance from the repository and creates target models.

4.3.2 Implementation Aspects

Transformation Technology

The choice of a suitable transformation technology is an important aspect of the implementation. The transformation technology comprises a transformation language and a corresponding execution environment. Section 2.2.3 gives a short overview of different paradigms for model transformations. The selection of a suitable transformation technology depends on various factors. One issue is the experience of a transformation developer. One further issue is access to meta-modeling environments. A transformation technology should provide direct access to models. Otherwise, development requires implementation of a costly interface. Often, general purpose programming languages (e.g. Java, C#, or JavaScript) offer interfaces to different meta-modeling environments. Special transformation languages (e.g. XSLT, ATL, or Epsilon) offer less support for different meta-model environments because these languages are specialized in a certain technology domain.

Interpreter versus Generator Approach

This aspect concerns the implementation of M1-transformations. There are two ways. The first way uses an interpretative approach. The M1-transformation implements a transformation against generic data structures for source and target models. This generic transformation is incomplete and requires the mapping between meta-model elements. The generic transformation reads the meta-model mapping (transformation instance of an M2-transformation) and applies each transformation rule. During the rule application, the meta-model mapping is passed to the current generic transformation rule. The rule analyzes the meta-model mapping, reads source model elements and creates corresponding target model elements.

The second way is the usage of a generator approach. The generator comprises a set of incomplete transformation rules. During the generation process, the generator reads the meta-model mapping (M2-transformation instance) and enriches each transformation rule with additional information. This information can be, for instances, type names of a source and target meta-model element. The generator creates an M1-transformation which is executable in a transformation system.

The advantage of the interpretative approach is that the transformation is executed in one step. The generator approach requires two steps, the generation of the transformation and the following execution of this generated transformation. For the implementation of the interpreter approach, the transformation languages must support the creation of model elements with a dynamic assignment to their meta-model elements. Not every transformation language supports this feature.
Furthermore, we assume that the complexity and management of the interpretative approach is higher than the generator approach. One reason is that the interpretative approach requires reflective functions for the processing of model data. This way of programming is difficult to debug and can easily lead to errors during run-time. Although the generator approach is also complex, the testing of generated code during design-time reduces the risk of potential errors.

Access Layer

The implementation of a bridge needs access to the model data. There are two ways: data interfaces and function interfaces. A data interface is often realized as a file export and import. An interface on the function layer is typically realized as an API. The advantage of an API in comparison to a data interface is that an API hides complexity concerning specific data constraints or additional logic. For instance, an API can offer methods for the creation of model elements. Hence, the usage of data formats often requires a more complex implementation because specific model constraints or logic must be re-implemented. However, the disadvantage of an API is that an API determines the transformation technology because an API is implemented in a certain programming language. The usage of a data format, in contrast, allows a broader technology spectrum because various programming or transformation languages support the general processing of data.

Transformation Instance

An M1-transformation needs a mapping between meta-model elements. This mapping is produced by a previously executed M2-transformation and is denoted as an M2-transformation instance. We identified two ways to describe such a transformation instance. The first option is the annotation of elements with labels in the target meta-model. A label references elements in the source meta-model and describes the relationship to these source elements. The second option is a separate description of relationships between source and target meta-model elements. This description can be stored, for instance, in a dedicated file. Each variant has its advantages and disadvantages.

4.3.3 Development Process

We suggest a process model for the development of M3Bs which is based on a classical waterfall model [Royce 1987]. The process can be divided into the following steps.

Analysis of meta-model environments The process starts with the analysis of both meta-modeling environments. The analysis comprises the complete model hierarchy including the meta-modeling language as well as the instance or conform to relationship between models, meta-models, and the meta-metamodel. Furthermore, the analysis should include interfaces for model data and possibilities to execute transformations. We suggest manuals
of the tool vendors and the analysis of the tool itself including user interface, programming interface and file interface.

**Bridge design** The second step is the design of the bridge. The design deals mainly with the decision of supporting the following features (see Section 4.2): direction, configuration, output and validation. Additionally, the design also includes technical aspects, such as integration layer and transformation technology.

**Definition of mappings** The third step is the definition of the M2-transformation and the definition of the corresponding M1-transformation. Often, there are various variants to map meta-modeling concepts. The step's result is the definition of mapping rules.

**Implementation and testing** The last step is the implementation of the bridge including all transformations and possible interface extensions. After the implementation, we suggest to test the bridge with a selected set of models. The test results can be used for a failure analysis and can serve as input for improvements.

Additional to the waterfall model, we suggest an incremental and iterative development. This extension is helpful because the information about structure and semantics of meta-metamodels and participating data structures is often unsatisfying. This often implies a trial-and-error development. Hence, we suggest to start with a small subset of mappings which can be gradually extended by further rules and features.

### 4.4 Combination of M3-Level-Based Bridges

#### 4.4.1 Combination Approach

An M3B is realized as a point-to-point connection between two meta-modeling environments. Thus, the connection of \( n \) environments would require the development of \( \frac{n(n-1)}{2} \) bridges. This amount of bridges can be reduced by the use of a star topology instead of a point-to-point topology. A star topology allows the reuse of bridges and requires only the development of \( n - 1 \) bridges. This decrease in bridges leads to a reduction of development and maintenance costs.

The development of a star topology needs a suitable meta-modeling environment as central element for the exchange of models between participating environments. Assuming there is a suitable environment for the representation of models, we propose an approach for the combination of bridges [Kern et al. 2014]. Figure 4.8 shows an overview of this approach.

One part is the meta-modeling environment which serves as central or common element. This environment is responsible for the representation of models and meta-models. Besides this central environment, there must exist a bridge between each meta-modeling environment and the central environment. In the presented case, there are two environments A and B, a central meta-modeling environment
4.4 Combination of M3-Level-Based Bridges

C, and two bridges between A and C and between B and C. It is possible to add another meta-modeling environment via a further bridge.

First step is the import of meta-models from environment A and B into the central environment C. In that case, the assumption is that environment A and B have a meta-model. If the meta-model does not exist in one of these environments, a developer must create the meta-model manually. Alternatively, a bridge can automatically create the meta-model in the target environment. In this case, the bridge from the source environment must support the export of meta-models and the bridge to the target environment must support the import of meta-models.

After the import of each meta-model into the central environment C, the next step is a meta-model alignment. We assume that the imported meta-models implement the same language. Despite the language-similarity, the imported meta-models have often heterogeneous structures. For instance, elements with the same meaning may have different names or relationships can be specified in a different way. In order to overcome this heterogeneity, a meta-model alignment is necessary. This alignment can be implemented as a transformation defining a mapping between both imported meta-models. This alignment transformation is specific to the participating meta-models.

After the definition of the alignment transformation, the first bridge (left) can import models from environment A into the central meta-model environment C. These models are the input of the alignment transformation. The execution of this transformation produces a target model which conforms to the second imported meta-model (from B). Afterwards, the second bridge (right) can export these models to environment B.
4.4.2 Alignment Transformation

The alignment transformation between the imported meta-models contains a simple mapping logic. The transformation often has a similar structure depending on the participating environments. Only the mappings between different meta-model elements differ from case to case. Development of this special kind of transformation is a repeating task and can be supported by a special mapping language. In contrast to the alignment transformation, the mapping language allows a more abstract description of correspondences between meta-model elements and hides execution details. Based on a defined mapping, a generator produces an executable transformation for the alignment. The generator is specific for the participating meta-model environments and transformation language. The upper part in Figure 4.8 presents this alignment approach with the mapping language and the generator approach.

The alignment approach aims to provide an abstract mechanism for specifying mappings without regarding the underlying meta-modeling environment. For this reason, we provide a language to define mappings and a generic tree representation for meta-models. The mapping is defined against this tree structure. The element tree structure is part of the mapping language and is presented in Figure 4.9. An element tree consists of an element container and a set of elements. An element container is a root node and represents a meta-model. Each element container has a name corresponding to the name of the imported meta-model. Furthermore, an element container has a binding type and a binding configuration. The binding type determines the responsible binding component and the interpretation of the binding configuration. Each element container comprises zero or more elements. Each element in a tree holds a reference to the original element in a meta-model. This reference is important in order to get specific details of the meta-model for the later

![Figure 4.9: Abstract syntax of the mapping language](image-url)
4.5 Summary

In this chapter, we presented the approach of M3-level-based bridges. This transformation approach is the central point of this thesis. At the beginning of this chapter, we defined the purpose and requirements for this approach. An M3B migrates models depending on their meta-models between different meta-model environments.

In the second part, we gave a formal description of the approach followed by an algorithmic description. A bridge comprises at least two transformations: the M2-transformation for meta-models and the M1-transformation for models. We characterized both transformation with the help of existing transformation properties. Furthermore, we described general features of an M3B, such as direction, configuration, output and validation.

In the third part of this chapter, we addressed different aspects for the development of bridges. We provided a sketch of an abstract transformation system for bridges and discussed selected implementation aspects. Afterwards, we presented a general development process.

The fourth and last part of this chapter described a way to combine bridges. The combination of bridges requires an alignment between meta-models. For this purpose, we presented an approach allowing the description of mappings between meta-models. A generator reads these mappings and produces corresponding alignment transformations.
5 Analysis and Mapping of Meta-Modeling Languages

In this chapter, we analyze selected meta-modeling environments and compare different meta-modeling languages. We derive an abstract framework of meta-modeling concepts and suggest different mappings between these concepts.

5.1 Structure of the Comparative Analysis

The development of an M3B requires the definition of mappings between different meta-modeling languages. An important aspect of these mappings is an understanding of different meta-modeling concepts. Hence, the purpose of this analysis is the investigation of meta-modeling languages and the extraction of typical meta-modeling concepts. Additional to these concepts, the implementation of M3Bs requires knowledge about technical details such as the structure of model data and the implementation of the conform to relationship. For this reason, the investigation also considers the analysis of data structures for models and meta-models. The result of this analysis is a concept framework of meta-modeling concepts which later can be used for the definition of mappings between meta-modeling languages [Kern et al. 2011].

The analysis considers a set of meta-modeling environments. Based on the study about interoperability in Section 3, we select a subset of tools which we want to investigate in more detail. In contrast to the interoperability study, we narrow down the number of tools for this analysis because a detailed investigation of each tool is quite labor intensive. Many tools have a black box character requiring reverse engineering of various interfaces and an inspection of other information sources (e.g. manuals, articles, or other documents). The tools in this study must fulfill the following three criteria.

Heavyweight meta-modeling The first criterion is the meta-modeling approach. There are the heavyweight and the lightweight approach. The lightweight approach adapts an already existing meta-model to domain-specific concepts. The heavyweight approach defines a completely new meta-model. The tool must support a heavyweight meta-modeling approach.

Graphical syntax The second criterion concerns the concrete syntax. There are graphical languages and textual languages. The tool must offer a graphical syntax with textual annotation in form of labels.
Availability The third and last criterion is the availability as a tool. The tools must be available, executable and usable without much effort. This allows a direct investigation and ensures the quality and plausibility of the results.

Based on these criteria, we select the following seven tools for the analysis: ARIS Business Architect, Cubetto Toolset, Eclipse Modeling Framework, Generic Modeling Environment, MetaEdit+, Microsoft Visio, and Modeling SDK for Visual Studio (MSDKVS).

The analysis uses the following procedure. We install each tool, investigate the user interface and define modeling languages. We apply modeling languages to create models. Through this step, we get an understanding of meta-modeling and modeling concepts. Parallel to this step, we study literature about the tool, for instance, manuals, tutorials, or API documents. Besides the user interface and the literature survey, we investigate data and programming interfaces. The advantage of the user interface is that it easily imparts an understanding of concepts but it is difficult to analyze relationships or an internal structure of concepts in detail. In contrast to the user interface, data and programming interfaces help to recognize the internal structure between meta-modeling and modeling concepts. The result of the analysis is a description of the meta-metamodel from each tool. Additional to this, we explain access mechanisms to model data. This is only necessary for the later implementation of M1-transformations.

Based on this analysis, we extract typical meta-modeling concepts and describe these concepts with possible variations. We use common terms for the description. The challenge is to find the right level of abstraction. On the one hand side, we must abstract from specific details in order to find commonalities. On the other hand side, we must consider specific details allowing the differentiation of concepts and variations. After the description of meta-modeling concepts, we show the occurrences of these concepts in the investigated meta-metamodels.

5.2 Analysis of Meta-Modeling Environments

5.2.1 Architecture of Integrated Information Systems

The Architecture of Integrated Information Systems (ARIS) is a framework for Business Process Management [Weske 2012] and enterprise modeling [Sandkuhl et al. 2014]. ARIS provides a methodical framework as well as tools. The ARIS method offers semi-formal and easily understandable modeling languages which enables the description of business organizations, their processes and underlying information technology in a holistic way. For this purpose, there are different views and abstraction levels [Scheer 2002]. ARIS supports a meta-modeling architecture and offers different modeling notations by default. ARIS supports a mixture of heavyweight and lightweight meta-modeling [Kern 2007a]. The ARIS tool vendor can create completely new languages. For instance, the entire ARIS modeling method is specified by exactly one meta-model. A user can adapt this meta-model for domain-specific purposes by using an ARIS filter. The filter mechanism is a lightweight meta-modeling approach.
We analyze the definition of languages, the filter mechanism and the modeling process. The result of this study is the ARIS meta-metamodel which is presented in Figure 5.1. Starting point is a filter which comprises all further language elements. A central element in a filter is a model type. A model type defines a set of models and references further elements: object types, connection types, symbols and lines. An object type defines a set of objects. A connection type defines a relationship between two or more object types. An object, connection, and model type can have attributes. This attribution is expressed by an attribute element. The range of attribute values is specified by a datatype which can be, for instance, boolean, date, float, integer or text.

On the model level, ARIS differentiates between element definitions and element occurrences. This concept enables the implementation of the ARIS view concept. An occurrence is a graphical representation of exactly one definition. An occurrence is connected to at least one or more occurrences. This view concept is also recognizable in the meta-metamodel with the elements on the left side and on the right side in Figure 5.1. Object occurrences and connection occurrences are defined by symbols and lines, respectively. Symbols and lines depend on a model type. Additional to this dependency, there are references from symbol to object type and from line to connection type. An assignment relationship between an object type and a model type allows the creation of links from objects to models on the model level. This assignment can be interpreted as a refinement or a detailed description of an object through another model.

Furthermore, we investigate the data structure of model data. Figure 5.2 illustrates the result of this analysis. The root element of an ARIS repository is a group. A group is a container for further model elements. A group can contain subgroups. This mechanism allows a hierarchical structure of models. The
ARIS view concept is also recognizable on the model level (see left and right side in Figure 5.2). A model is a graphical representation and builds a view on definition elements. A model is implemented as a canvas and contains occurrences. Thus, a model is similar to a diagram. An occurrence is a graphical representation of exactly one definition. A definition is connected with at least one or more occurrences. There are three types of model elements: objects, connections and attributes. These three types support the definition-occurrence concept. Object definitions and object occurrences are entities which can be connected by connection definitions and connection occurrences, respectively. Connection definitions, object definitions and models can own attribute values. An attribute definition can also have a graphical representation in a model. This representation of attribute definitions is implemented by attribute occurrences. Models, object definitions, connection definitions, object occurrences and attribute definitions are typed elements. These model elements reference meta-model elements by the usage of the type property.

5.2.2 Cubetto Toolset

Cubetto is another modeling tool in the domain of business process modeling. In contrast to ARIS, which is suitable for large organizations, Cubetto addresses mainly small and medium-sized organizations. The tool allows the definition of
modeling languages and offers pre-defined languages, such as BPMN, EPC, OOA, and UML. The meta-modeling language in Cubetto is denoted as E3-method. The development of Cubetto started at the University of Dresden and was part of the thesis from Greiffenberg [2003]. A language engineer can define a language by using dialog-based wizards. Afterwards, the tool configures a generic modeling component with the previously defined language. The meta-metamodel of Cubetto is presented in Figure 5.3. An investigation of the model data structure is unnecessary because we do not implement a bridge with Cubetto.

The definition of a meta-model in Cubetto starts with the creation of a package. A package defines a namespace and is a container for model types. A model type defines a set of models. A model type includes object types. An object type defines a set of objects. Objects are model elements which are represented graphically as symbols in a model. An object type can have properties. The co-domain of properties can be a simple data type (e.g. string, integer, etc.) or other object types that already exist in a meta-model. This reference of object types enables the definition of relationships between object types. Furthermore, properties support a cardinality and the specification of a structure type (e.g. list, set, or bag).

Similar to ARIS, Cubetto supports a view concept. A model type includes a view which in turn contains graphical representations. A presentation contains graphical representations of object types. An object presentation contains a set of property presentations. A property presentation defines a view on properties. An object type and property can have more than one presentation but a presentation belongs to exactly one object type or property. E3 supports single inheritance between object types.
5.2.3 Eclipse Modeling Framework

The Eclipse Modeling Framework (EMF) [Steinberg et al. 2009] is a central project in Eclipse facilitating the implementation of data models for Eclipse applications. EMF supports the specification of data models and the generation of data structures. The framework has a great importance within the Eclipse community because many Eclipse-based applications use EMF. Based on EMF, there is a variety of tools for building a tool infrastructure supporting the Model-Driven Development paradigm. The core artifact of EMF is Ecore which is the specification language for data structures. Regarding the Model-Driven Development paradigm, Ecore is considered as a meta-metamodel. A data structure conforming to Ecore is considered as a meta-model and data conforming this data structure is considered as a model.

A minimal subset of Ecore is illustrated in Figure 5.4. Starting point of an Ecore meta-model is the creation of an EPackage. An EPackage can contain other EPackages and EClasses. An EPackage has a unique identifier and realizes a namespace concept which prevents name conflicts and offers the creation of a logical structure for meta-models. Another important concept in Ecore are EClasses. Similar to a Java class, an EClass describes a set of instances sharing same features. These instances are at the model level. An EClass can have two kinds of features (EStructuralFeature): EReferences and EAttributes. EAttributes allow the definition of attributes. The co-domain of an attribute is defined by an EDataType or an EEnum. Ecore supports common datatypes, such as string, integer, float or date. EReferences enable the definition of relationships between EClasses. The co-domain of an EReference is an EClass. EReferences have a cardinality and allow the definition of an inverse EReference. An inverse EReference defines the opposite reference to the original EReference. EClasses support inheritance, more precisely, multiple inheritance. The inheritance allows for reuse of EReferences and EAttributes in a class hierarchy.

The data structure of EMF models corresponds directly to a defined EMF metamodel. EMF creates an equivalent Java class with corresponding attributes for an

Figure 5.4: Ecore – Meta-metamodel of Eclipse Modeling Framework
given EClass. That is, the instantiation of an EClass creates a Java object which
is an instance of the created Java class. Additional to this meta-model-specific
data structure, EMF support a generic and reflective mechanism for models. An
EClass is inherited from EObject. EObject offers methods to query, for instance,
the EClass of an EObject and to query values from an EAttribute or EReference.

5.2.4 Generic Modeling Environment

The Generic Modeling Environment (GME) is a meta-modeling tool developed by
the Institute for Software Integrated Systems at the Vanderbilt University [Ledeczi
et al. 2001; ISIS 2006]. GME is primarily used for domain-specific modeling in the
area of electrical engineering. GME is divided into two parts: a tool for language
definition, called MetaGME, and a tool for using languages, called GME. A lan-
guage definition includes meta-models and a description of the visualization. We
only analyze the meta-modeling language and not the data structure because we
do not implement a bridge with GME.

The analyzed meta-metamodel is presented in Figure 5.5. Starting point for
a language is the creation of a paradigm. A paradigm comprises model types. A
model type defines a set of models. Furthermore, a model type can contain concepts
inherited from First Class Object (FCO). A model is represented as a graphical
canvas. Alternatively, a model can be represented as a symbol on a canvas. In this
case, the symbol holds a reference to a model. A double click on this symbol opens
the underlying model. This concept allows a hierarchical structure of models and
enables a refinement of models. A model type can include atoms. An atom defines

![Figure 5.5: Meta-metamodel of Generic Modeling Environment](image-url)
a class of objects. These objects represent entities or nodes on a canvas and can have connections to other objects. Besides atoms, GME offers three different ways to define relationships between atoms: connection, reference and set. The first way is a connection between two atoms. A connection is similar to a binary association and is graphically represented by a line between two atoms. A connection has roles defining the participation of an atom or model in a connection. The second way is a reference between elements. In contrast to connections, references have no graphical representation in a model. A reference is realized as a value or pointer in an attribute field. The third way is the definition of a set relation. The graphical visualization of a set relation allows the user to show or hide elements connected to a selected element. Each subclass of FCO (atom, set, reference, connection, and model) can have attributes. Furthermore, GME supports inheritance between meta-modeling elements.

5.2.5 MetaEdit+

MetaEdit+ is a meta-modeling tool for software and system development, especially Domain-Specific Modeling (DSM) or Model-Driven Development (MDD). MetaEdit+ allows the definition of modeling languages, offers a generator engine for code generation, and provides a model repository. We analyzed the meta-modeling language and the storage structure for models. For this purpose, we studied the following literature: [Kelly and Tolvanen 2008; Tolvanen 1998; Kelly 1997]. Additionally, we investigated the tool itself, the user interface, programming interface and the XML export format.

Figure 5.6 shows the meta-metamodel in MetaEdit+. The meta-metamodel is denoted as GOPPRR. This abbreviation stands for the main concepts: Graph, Object, Property, Port, Role and Relationship. A graph type defines a set of graphs. A graph is represented as a canvas. A graph type consists of object types and relationship types. An object type defines a class of objects. An object is represented as a symbol on a canvas. Objects can be connected by relationships. The definition of relationships is realized by a binding concept. A binding allows the creation of a complex relationship. A binding comprises a relationship type and a combination of role types, port types and object types. A role type in a binding describes the participation of object types in a relationship. The number of included role types defines the arity of a relationship. GOPPRR supports the definition of binary and n-ary relationships. A port type is a constraint which restricts the possible connections between relationships and objects to specific locations of the participating objects.

The language concepts: graph, object, port, role and relationship type can have attributes. Attributes are realized by properties. GOPPRR supports single values or a collection of values. The co-domain can be common data types (e.g. string, text, number, boolean) as well as references to other meta-model elements: graph, object, port and relationship types. This reference enables the definition of a relationship to other modeling elements which is not represented graphically in a model but as an attribute value. GOPPRR supports inheritance between language
5.2 Analysis of Meta-Modeling Environments

Figure 5.6: GOPPRR – Meta-metamodel of MetaEdit+

elements. This enables the reuse of properties and other constraints. Furthermore, there are special relations between meta-model elements. There is a decomposition relation between object types and graph types. A similar concept is the explosion that allows a relation between object, relationship, roles types and graph types. These special relationships enable the refinement of model elements.

The data structure for models follows the structure for meta-models. To get this structure, we analyzed the API and the XML schema for models. Model data is mainly stored as a set of objects. These objects are instances of the type MEOop. The meta-model element of a model element is represented as METype. The API offers different methods for reading and creating MEOop objects. For instance, the method MEOopArray objectSet(MEOop receiver) expects a graph object as input and returns a set of all objects in this graph. The general structure of model data is as follows. The entry point is a project. A project contains graphs and a graph contains objects, relationships, roles and bindings. The binding object offers further methods to read and write objects relating to a binding.

5.2.6 Modeling SDK for Visual Studio

The Modeling SDK is part of Visual Studio and allows the definition and usage of domain-specific languages. The former name of this SDK was Microsoft Domain-Specific Language Tools. The SDK includes a code generator engine T3/T4 allowing the generation of source code. For the analysis of the meta-metamodel, we used literature (e.g. [Cook et al. 2007]), the user-interface and the programming
interface of the tool. We only analyzed the meta-modeling language, skipping over the model data structure because we do not implement an M3B with this tool.

The result of the analysis is illustrated in Figure 5.7. Starting point is a language. A language is a container for all language elements and defines a namespace. A language consists mainly of domain classes. A domain class defines a set of modeling elements which can be interpreted as objects or symbols on a canvas. The Modeling SDK also supports the definition of relationships. A relationship is defined by using a domain relationship and is represented in a model as a line between objects. A domain relationship has two domain roles which reference a source and target domain class, respectively. A role describes the participation of a domain class in a domain relationship. Furthermore, a domain class can have attributes. An attribute is defined by a domain property. As the domain relationship is a sub class of the domain class, domain relationships can also have attributes. MDSDK supports single inheritance between classes and relationships. This inheritance allows for reuse of attributes.

5.2.7 Microsoft Visio

Microsoft Visio is a universal modeling and data visualization tool. Visio is one of the most commonly used modeling tools [Loos and Fettke 2007]. The applications of Visio are manifold. Typical areas include, for instance, software engineering, database modeling, or technical drawings. Similar to other meta-modeling tools, Visio has no explicit description of its meta-metamodel. For the analysis of the Visio meta-metamodel, we used the tool itself, the user interface and programming interface (Visio object model), and available literature (e.g. Biafore [2007]).

The meta-metamodel of Visio is illustrated in Figure 5.8. Visio allows the definition of languages by using stencils. A stencils can be regarded as a meta-model. A stencil contains a set of masters. A master can be divided into two types: object master and connection master. An object master defines a set of model entities. These entities are also denoted as shapes and are represented as symbols on a can-
A connection master is a relation between two object masters. An instance of a connection master is represented in a model as a line between shapes. Object and connection masters can have properties. Each property has a datatype defining the co-domain of possible values. A master element can be a composition of other masters.

The data structure for Visio models is presented in Figure 5.9. A Visio model is stored in a file and is denoted as document. A document consists of pages. A page is a drawing canvas and has a name. A page can contain shapes. A shape is a model element which is an instance of a master. The relationship between shapes and master is realized by a type reference of a shape. A shape can be a two-dimensional element or a one-dimensional shape. The first kind of shapes relates to object masters and is similar to nodes in graphs. The second kind of shapes relates to connection masters and is similar to edges in a graph. Additionally, the Visio object model allows nesting and grouping of shapes. This enables the creation of complex shapes. Shapes can have attributes. These attributes can be defined in a master or a shape can have attributes which are defined for this shape only on the model level. An attribute has a name, a value and a data type. Furthermore, all shapes have a default attribute which is named text. Additionally, shapes have further properties describing the graphical style of shapes. These properties are accessible via the shape sheet reference.
5.3 Comparison of Meta-Modeling Concepts

5.3.1 Basic Concepts

Based on the investigation of various meta-modeling languages, we derive basic meta-modeling concepts. The definition of these concepts requires knowledge about general modeling concepts. These modeling concepts and a relation to their meta-modeling concepts are presented in Figure 5.10. This figure is divided into two parts: a model level and a meta-model level. The model level shows examples of common model concepts such as objects and relations. The meta-model level comprises different meta-model concepts such as object type and relation type. The occurrence of a meta-modeling concept in each language is described in Table 5.1. The left side shows the abstracted meta-modeling concepts and the columns on the right side describe the concrete realization of each concept in a meta-modeling language.

![Figure 5.10: Basic meta-modeling concepts](image)

**Object type**

An object is a discrete identifiable entity in a model. In graph theory an object can be regarded as a node in a graph. Figure 5.10 shows two objects which are depicted as rectangles (Object A and Object B). Depending on the specific language, objects are depicted by specific symbols or icons. An object type defines a set or a class of objects with equal features. These features can be, for instance, relationships to other elements or attributes. An object relates to an object type on the meta-
5.3 Comparison of Meta-Modeling Concepts

<table>
<thead>
<tr>
<th>Object type</th>
<th>ARIS</th>
<th>Ecore</th>
<th>GOPPRR</th>
<th>GME</th>
<th>MSDKVS</th>
<th>MS Visio</th>
<th>E3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relation type</td>
<td>Object, Symbol</td>
<td>EClass</td>
<td>Object</td>
<td>Atom</td>
<td>Domain Class</td>
<td>Object Master</td>
<td>Object</td>
</tr>
<tr>
<td>Attribute</td>
<td>Connection, Line</td>
<td>EReference</td>
<td>Relation, (Collection)</td>
<td>Connection, Set, (Reference)</td>
<td>Domain Relationship</td>
<td>Object</td>
<td>Property</td>
</tr>
<tr>
<td>Model type</td>
<td>Model</td>
<td>–</td>
<td>Graph</td>
<td>Model</td>
<td>Domain Property Language</td>
<td>–</td>
<td>Model</td>
</tr>
</tbody>
</table>

Table 5.1: Basic meta-modeling concepts

model level. An object has exactly one object type and an object type can have more than one object.

The realization of an object type in each environment is described in Table 5.1. Each language supports the definition of object types. ARIS with its view concept allows the definition of different views. This view concept leads to the distinction between two sorts of object types. ARIS provides object types for object definitions and types for object occurrences. The second sort of object types is represented by symbols in ARIS. Ecore uses EClasses, GOPPRR uses object types, GME offers the atom concept, MSDKVS provides domain classes, Visio supports (object) masters, and E3 offers object types.

Relation type

Objects can be connected by relations. In graph theory a relation can be regarded as an edge in a graph. Figure 5.10 depicted a relation as a black line with an arrow and a circle in the middle. A relation type defines a set or a class of relations with equal features. These features can be, for instance, source and target object type or attributes. A relation type connects object types and is defined as a subset of the (n-ary) Cartesian product over the participating object types. Analogous to an object, a relation is an instance of a relation type. A relation has exactly one relation type and a relation type can have more than one relations. A relation type is characterized by the following features.

Arity Arity specifies the number of object types that can be involved in a relation type. There are two possibilities: binary or n-ary. A binary relation is defined as cross product between two object types and an n-ary relation type is defined as cross product between n object types.

Multiplicity Multiplicity defines the number of objects that can be related to another object in the context of a relation. Multiplicity can be defined with concrete values or intervals (e.g. n, 0..1, 0..n, 1..n). If multiplicity is not supported, then the default value is 0..n (n ∈ N).

Composition Some meta-modeling languages support the creation of a composite relation type. A composition influences the life-cycle dependency between a
container object and contained objects. Generally, if the container object is deleted, every related/contained object is deleted.

**Object-set** Normally, a relation type assigns only one object type for each relation type end. The object-set feature enables the connection of a set of object types for each end. This means, a relation type is defined as a cross product between two or more sets of object types.

**Role** A relation type can support a role concept. A role defines the participation of an object in a relation. Thus, a role concept allows a detailed description of a connection between a relation and an object. Figure 5.10 depicted a role as part of a relation. There are two options for roles. The first option for a relation type is to support by default roles which are implicitly defined with the creation of a relation type. These roles are denoted, for instance, as source and target role. The second option is the support of role types which can be defined by a dedicated type element. A role type defines a set or class of roles with properties such as participating relations, objects and attributes. If a meta-metamodel supports a role concept, then a relation points to an object via a role. This leads to the effect that some relation type features (e.g. multiplicity or object-set) are transferred to role types.

**Port** A port is a special connection point between a relation and an object. A port defines a specific anchor point for a relation and restricts the connection possibilities between a relation and objects. Thus, a port allows the expression of additional semantics for the participation of objects in a relation. A port type defines a set or a class of ports with the same features. Figure 5.10 depicted a port as rectangle beside objects (port *In* and *Out*).

| Arity   | Multiplicity | Composition | Object-set | Role     | Multiplicity | Object-set | Port   | ARIS    | Ecore    | GOPPRR   | GME     | MSDKVS   | MS Visio | E3      |
|---------|--------------|-------------|------------|----------|--------------|------------|--------|---------|----------|---------|---------|---------|---------|---------|---------|
| binary  | default      | –           | –          | –        | –            | –          | –      | 0..n    | 0..n     | 0..n    | default | 0..n    | 0..n    | –       | –       |
| n-ary   | binary       | –           | –          | –        | –            | –          | –      | –       | –        | –       | –       | –       | –       | –       |
| binary  | binary       | –           | –          | –        | –            | –          | –      | –       | –        | –       | –       | –       | –       | –       |
| binary  | –            | –           | –          | –        | –            | –          | –      | –       | –        | –       | –       | –       | –       | –       |
| binary  | –            | –           | –          | –        | –            | –          | –      | –       | –        | –       | –       | –       | –       | –       |

Legend: • supported • not supported – impossible

Table 5.2: Features of relation types
5.3 Comparison of Meta-Modeling Concepts

All investigated meta-modeling languages support the definition of relation types. Table 5.2 shows the relation features in each meta-modeling language. Some languages provide more than one way for the definition of relations. Each meta-modeling language supports the definition of binary relations, except GOPPRR which even offers n-ary relations. Furthermore, all meta-modeling languages, except ARIS and Visio, allow the definition of a multiplicity. ARIS and Visio offer a default multiplicity between zero and endless. Ecore, GME, and MSDKVS support the composition feature. The object-set feature is only supported in ARIS and GOPPRR. GOPPRR, GME, and MSDKVS offer roles. The support of roles leads to the transfer of the multiplicity and object-set feature. The port concept is only supported in GOPPRR and GME.

Variants of Relation Types

Additional to the identified features for relation types, there are five typical relation variants which depend on a certain combination of features. These relation variants are illustrated in Figure 5.11. The variants which are supported in a particular meta-modeling language are listed in Table 5.3.

Reference-relation A simple solution for a relation is a reference-relation. A reference can be interpreted as a pointer from one object to another. A reference-relation is binary, often depends directly on an object type, and is not attributable. A reference can be implemented as an object attribute which holds unique identifiers of target objects. Figure 5.11(a) illustrates a reference-relation. EMF and E3 support only references. GME and GOPPRR support references as one possibility among many.

Binary object-relation A binary object-relation uses a dedicated element for a relation. This relation element references source and target objects. Figure 5.11(b) shows a binary object-relation. In contrast to a reference which has a strong dependency on source and target objects, the object-relation can be considered as a “stand-alone” modeling element which can exist without source and target object. This relation is often attributable because the dedicated relation element can have attributes. All investigated meta-modeling languages, except EMF and E3, support this variant.

N-ary object-relation The n-ary object-relation in Figure 5.11(c) is analogous to a binary-object relation with the additional feature that this relation type has more than two relation ends and thus can reference more than two object types. Only GOPPRR allows the definition of n-ary relations.

Object-set relation An object-relation (binary and n-ary) references exactly one object type for each relation end. In contrast to this, the object-set relation in Figure 5.11(d) enables the connection of a set of object types for each relation end. The interpretation of this relation is the Cartesian product between all participating object sets. For instance, the relation in Figure 5.11(d) allows the instantiation of relations between objects which are instances of the object
types O1 or O2 as source and O3 or O4 as target. ARIS and GOPPRR offer object-set relations.

**Role-relation** An object-relation consists of a relation element with a source and target reference. These two references can be regarded as implicit roles because these roles are realized by references with a strong dependency on their relation element. In contrast to this, there is a relation which supports an explicit role concept (see Figure 5.11(e)). The participating roles are realized as a “stand-alone” model element. Additionally, explicit roles can have their own identifier (name) and are attributable. GOPPRR and GME support role relations.

![Diagram of relations](image)

Figure 5.11: Typical variants of relations

<table>
<thead>
<tr>
<th></th>
<th>ARIS</th>
<th>Ecore</th>
<th>GOPPRR</th>
<th>GME</th>
<th>MSDKVS</th>
<th>MS Visio</th>
<th>E3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference-relation</td>
<td>○</td>
<td>•</td>
<td></td>
<td>○</td>
<td>•</td>
<td></td>
<td>○</td>
</tr>
<tr>
<td>Binary object-relation</td>
<td>•</td>
<td>○</td>
<td>•</td>
<td>○</td>
<td>•</td>
<td>•</td>
<td>○</td>
</tr>
<tr>
<td>N-ary object-relation</td>
<td>○</td>
<td>○</td>
<td>•</td>
<td>○</td>
<td>○</td>
<td></td>
<td>○</td>
</tr>
<tr>
<td>Object-set relation</td>
<td>•</td>
<td>○</td>
<td>•</td>
<td>○</td>
<td>○</td>
<td></td>
<td>○</td>
</tr>
<tr>
<td>Role-relation</td>
<td>○</td>
<td>○</td>
<td>•</td>
<td>•</td>
<td>○</td>
<td></td>
<td>○</td>
</tr>
</tbody>
</table>

Legend: ■ supported ○ not supported ● limited supported

Table 5.3: Support of relation variants
5.3 Comparison of Meta-Modeling Concepts

Model Type

Generally, a model is an instance of a meta-model and conforms to a meta-model. However, some meta-modeling environments extend this meaning. Often a model is a graphical view or diagram of model elements. There is no difference between the model itself and the view on said models. Depending on a concrete meta-modeling environment, a model often serves as container for objects, relations and other model elements. Furthermore, a model is often represented as a canvas (see Figure 5.10). A model type defines a set or class of models with equal properties. Model types reference other types, such as object or relation types.

Links

Some meta-metamodels support the definition of links between a meta-model element and a model type. This link concept allows a detailed description of a model element (e.g. objects or relations) by using other models. This refinement approach is important for the combination and integration of different models.

ARIS, GOPPRR, GME, E3 and MSDKVS support model types. In these tools, the instantiation of a model type creates a model with elements conforming to type elements contained in this model type. Visio does not support model types but allows the creation of models. A Visio model can contain elements which are instances of masters in an arbitrary stencil. Ecore does not support model types and models. Regarding the link concept, ARIS uses the link concept to support the ARIS view concept. GOPPRR also offers the link feature under the names: “explosion” and “decomposition”. GME offers the possibility to define models as objects. A click on an object opens a corresponding model.

Attribute

An attribute defines a property for model elements. On the model level, an attribute can hold values. Attributes can be represented as text labels in a model (see Figure 5.10). Attributes can have the following features.

Data type A data type defines the co-domain of an attribute. Attributes can support simple data types (e.g. integer, string, or float) and complex data structures (e.g. list, bag, set). Furthermore, attributes can use other meta-model elements as their co-domain. This allows the definition of references between elements. In this case, an attribute become a relation type (see Section 5.3.1, particularly reference-relation).

Multiplicity This feature defines the amount of values or elements which can be stored in an attribute. There are single-values for one value and multi-values for more than one values. Additionally, multiplicity can define either that an attribute is optional or mandatory. Possible multiplicity values are: 1, 1..n, 0..n or n with n ∈ N.

Ordered If the multiplicity allows multi-values, values can be sorted in a defined order.
Table 5.4: Features of attributes

**Unique** If the multiplicity allows multi-values and an attribute is marked as unique, no value may occur multiple times.

**Default value** This feature allows the definition of a default value which is set after the instantiation of a model element.

**Attributable** This feature describes the elements which are attributable.

All investigated meta-modeling languages support the definition of attributes. Table 5.4 shows the support of the attribute concept in each meta-modeling language. ARIS, GME and MSDKVS only support single-value attributes. Ecore, GOPPRR, Visio and E3 allow the definition of multi-value attributes. The majority of languages supports an order, uniqueness and a default value. Each language supports simple data types. Additional to this, GOPPRR and E3 allow other meta-model elements as data types.

Each meta-modeling language supports the attribution of object types. Furthermore, ARIS, GOPPRR, GME, MSDKVS and MS Visio enable the attribution of relation types. Relation types in Ecore and E3 cannot have attributes because both languages use references which are not attributable. Furthermore, GOPPRR supports the attribution of role and port types. ARIS, GOPPRR and GME offer the attribution of model types. Additional to the attribution on the meta-model level, Visio supports the individual attribution of elements on the model level.

### 5.3.2 Additional Concepts

**Inheritance**

Many meta-modeling languages support inheritance. Inheritance is a special relation between meta-model elements. The concrete semantics of inheritance depends
5.3 Comparison of Meta-Modeling Concepts

<table>
<thead>
<tr>
<th></th>
<th>ARIS</th>
<th>Ecore</th>
<th>GOPPRR</th>
<th>GME</th>
<th>MSDKVS</th>
<th>MS Visio</th>
<th>E3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inheritance</td>
<td>–</td>
<td>multiple</td>
<td>single</td>
<td>multiple</td>
<td>single</td>
<td>–</td>
<td>multiple</td>
</tr>
<tr>
<td>Inheritable concepts</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Object type</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Relation type</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Role type</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Port type</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Model type</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Structuring</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Filter, Model type</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EPackage</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project, Graph type</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Folder, Paradigm</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Language, Namespace</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stencil Package</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraints</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OCL</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>proprietary</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OCL dialect</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPL</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPL</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Legend: ● supported ○ not supported – impossible

Table 5.5: Additional concepts

on the interpretation in a specific meta-modeling environment. However, inheritance often allows the reuse of attributes, constraints, or behavior of elements. We found the following inheritance features.

**Participating elements** This feature defines meta-model elements which can be part of an inheritance relation. These elements are, for instance, object, relation, role, or model type.

**Single vs. multiple inheritance** Single inheritance is when a (sub)element inherits from one (super)element. Multiple inheritance is when a (sub)element can inherit from more than one (super)elements.

Five out of seven meta-metamodels support inheritance. Table 5.5 shows the support of inheritance. GOPPRR and MSDKVS support single inheritance. Ecore, GME and E3 support multiple inheritance. Each language supports at least inheritance of attributes. Some languages support inheritance of additional modeling constraints. All seven meta-metamodels enable inheritance between object types. Additionally, GOPPRR and GME realize inheritance between relations and models. Furthermore, GOPPRR supports inheritance between roles and ports. ARIS and MS Visio do not support any kind of inheritance.

**Structuring**

There are a number of concepts for structuring meta-models into logical units. Structuring concepts are important, for instance, to handle a set of meta-models or large meta-models. Structure concepts often enable the creation of namespaces, the combination of different meta-models and the reuse of meta-model elements. Generally, structuring concepts on the meta-model level have no direct impact on the model level.

Table 5.5 shows structuring concepts in each meta-modeling language. ARIS supports filters containing a set of selected and adapted modeling elements. Additionally, ARIS supports the concept of a model type which is a logical unit for object
types, relation types, symbols and lines. Ecore supports EPackage packages which allow the definition of namespaces for EClasses. GOPPRR supports projects containing language elements. Furthermore, GOPPRR supports graph types referencing further language elements. GME supports folders containing language elements. A paradigm references these language elements and defines a modeling language. MSDKVS supports the creation of a language element containing further elements (mainly domain classes). Visio uses stencils containing masters. E3 supports a package concept.

**Constraint Language**

Constraints allow the definition of additional conditions for meta-models which have to be fulfilled during or after modeling. There are different mechanisms for the definition of constraints. One option is the usage of an additional language. There are special constraint languages (e.g. Object Constraint Language (OCL) [OMG 2014b]) or general purpose languages (GPL). Another way is the usage of proprietary solutions (e.g. dialog-based definition of constraints). The mechanisms differ in syntax, semantics and expressiveness.

All meta-modeling languages except ARIS and E3 allow the definition of constraints (see Table 5.5). GOPPRR supports a proprietary mechanism, Ecore supports OCL, and GME uses an OCL dialect. Visio and MSDKVS support a general-purpose programming languages for defining constraints.

**5.3.3 Further Observations**

**Model Conformity**

The M3B approach requires a classical model hierarchy consisting of three model levels (see Section 2.1.4). This model hierarchy allows only the creation of models which are strictly conform to a meta-model. Such a meta-model defines the complete set of possible models. It is forbidden by the conform to relationship to create models without a corresponding element in a meta-model. For instance, EMF allows a strict conform to relationship. Additional to this strict conform to relationship, we recognized models which do not conform strictly to a corresponding meta-model. In this case, models can contain elements not defined in a meta-model. For instance, Visio allows the creation of properties which are individually assigned to a model element. These properties are not part of a master (meta-model element). This possibility gives more flexibility during modeling but the processing of models is more complex and the quality of models cannot be ensured by a meta-model. For the transformation of models, the strict conform to relationship is an advantage because the transformation must only consider the corresponding meta-model. The support of a non-strict conform to relationship requires more effort because the transformation must consider additional model elements. The handling of these additional elements makes a transformation more complex.
5.3 Comparison of Meta-Modeling Concepts

Data Structure of Models

The transformation of models and meta-models requires knowledge about the underlying data structure for reading and writing models. The data structure for meta-models is often equal to the structure of the meta-metamodel in a model environment. That is, there is one data structure for storing all possible meta-models. Based on this data structure, a meta-modeling environment can implement an API for meta-models.

Regarding the data structure for models, there are two possibilities. The first possibility is a data structure equal to the structure of concrete meta-models. This direct dependency requires the creation of a specific data structure for each meta-model. This specific data structure can only store models conforming to a concrete meta-model. For instance, EMF allows the generation of Java classes. These Java classes are equivalent to the meta-model. The instantiated objects of these generated Java classes represent model data. The second possibility is the usage of a generic data structure for all models independent of a specific meta-model. This data structure follows the structure of the meta-metamodel instead of a specific meta-model. The meta-model environment must ensure the conformity between models and meta-models. For instance, ARIS and Visio implement a generic data structure.

One further variant is the usage of a common data structure for models and meta-models. In this case, a meta-modeling environment uses one generic data structure. For instance, a generic graph format allows the storage of models and meta-models as graphs. Additional application logic must ensure the creation of correct models and meta-models.

The storage mechanism is interesting for the implementation of an M3B. Using a specific data structure for each meta-model leads to specific transformation rules in an M1-transformation. The automatic creation of specific rules for the M1-transformation can be implemented as a generative approach. A generic data structure allows the implementation of a generic M1-transformation which must consider specific meta-model elements during run-time. The generic M1-transformation can use an interpreter approach. Both implementation approaches are described in Section 4.3.2.

Separation of Abstract and Concrete Syntax

Generally, there should be a strict separation between abstract and concrete syntax. This separation eases the definition of languages and enables reuse of language elements. Abstract syntax is defined by a meta-model and concrete syntax is described by additional artifacts such as icons, symbols, SVG graphics.

Some tools differentiate strictly between concrete and abstract syntax. For instance, MetaEdit+ or GME have a clean separation of syntax. In contrast, there are tools facilitating no strict separation of abstract and concrete syntax. For instance, ARIS and Visio mix abstract and concrete syntax. ARIS supports symbols. On the one hand, a symbol can be regarded as part of the abstract syntax because a symbol can be used for the creation of model elements (object occurrences). On
the other hand, a symbol can be regarded as part of the concrete syntax because a
symbol is represented graphically in a model as icon. Visio uses masters as part of
the abstract syntax and each master includes a graphical description as part of the
concrete syntax. A change of the concrete syntax influences the master definition.

The separation of abstract and concrete syntax eases reverse engineering of meta-
metamodels and the implementation of transformation rules. A cloudy separation
makes reverse engineering of a meta-metamodel difficult. This uncertainty can
have negative effects on the implementation of M3Bs.

5.4 Mapping of Meta-Modeling Concepts

After the extraction of typical meta-modeling concepts, we present mapping rules
between these concepts and discuss their advantages and disadvantages. The map-
ing between meta-modeling concepts is a challenging task due to their complexity.
The complexity results from the number of combinations between different concepts
and their variants. Formally, we must consider all possible mappings between all
meta-modeling concepts and their variations. This means, the mapping discussion
must include the cross product of all concepts. This would be not reasonable and
beyond the scope of this work. To handle the complexity, we describe reasonable
mappings. Firstly, we present mappings between basic concepts. Afterwards, we
describe mappings between relation types, attributes and further concepts. The
mapping rules are described in tables. A rule consists of a source and target cell
and a description of the mapping rule. Each mapping rule has a unique identifier.

5.4.1 Basic Mappings

We present mappings between basic concepts in Table 5.6. Please note, we do
not consider each combination of mappings. For instance, we do not discuss a
mapping from object type to relation type because we assume that each meta-
modeling environment supports object types. Hence, there is no reasonable need
for a mapping from object types to relation types.

<table>
<thead>
<tr>
<th>ID: R1</th>
<th>Source: object type</th>
<th>Target: object type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Each language supports object types. Therefore, a source object type can be mapped to a target object type including all attributes (see rule R4).</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID: R2</th>
<th>Source: relation type</th>
<th>Target: relation type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Similar to object types, all meta-modeling languages support relation types. Therefore, a source relation type should be mapped to a target relation type including its referenced object types (see rule R1) and possible attributes (see rule R4). Relation types have different features. The mapping of these features is discussed in the next Section 5.4.2.</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.6: Mapping of basic concepts – continued on next page
5.4 Mapping of Meta-Modeling Concepts

<table>
<thead>
<tr>
<th>ID: R3</th>
<th>Source: relation type</th>
<th>Target: object type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normally a source relation type should be mapped to a target relation type (see rule R2), but if the source meta-model supports attributable relation types and the target meta-model supports relation types without attributes, rule R2 would lead to an information loss. In this case, one mapping option is the transformation of a relation type into an object type which is attributable. The transformation must consider the referenced object types of the source relation type. These references must be transferred to the created target object type. This can require the creation of new relation types in the target meta-model. This mapping option preserves the attribution of a relation type. Nevertheless, the relation type is encoded as object type. This must be considered for later interpretation of this special object type in the target meta-model.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID: R4</th>
<th>Source: attribute</th>
<th>Target: attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Each meta-modeling language supports the attribution of concepts. Therefore, an attribute of a source concept should be mapped to an attribute of a corresponding target concept. Attributable concepts are, for instance, object or relation types (see Table 5.4). Attributes have different features requiring a more detailed description of this rule in Section 5.4.3.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID: R5</th>
<th>Source: model type</th>
<th>Target: model type</th>
</tr>
</thead>
<tbody>
<tr>
<td>If both meta-modeling languages support model types, a source model type can be mapped to a target model type. References to other concepts should be included in this transformation. This can require the application of rule R1 for object types, R2 for relation types and R4 for attributes.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID: R6</th>
<th>Source: model type</th>
<th>Target: object type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normally, a model type should be transformed into a model type but some meta-modeling languages offer no model type. In this case, model types in the source meta-model can be mapped into object types in the target meta-model. The referenced elements (object or relation types) of the source model type must be transferred to the target meta-model. This requires the creation of additional relation types.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID: R7</th>
<th>Source: object type</th>
<th>Target: model type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Under certain circumstances, an object type can be mapped to a model type. An object type can connect other object types by a containment relationship. In this case, the container object type can be transformed into a model type. All contained object types must be referenced in the created model type.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Table 5.6: Mapping of basic concepts – continued from previous page |

5.4.2 Mapping of Relation Type

In this section, we describe mappings between relation types in dependency of their features. The description is based on rule R2 from Table 5.6. The marked rectangles in the feature row show the current configuration of the source and target relation type.
## A arity

<table>
<thead>
<tr>
<th>ID: R2/1/1</th>
<th>Feature: arity</th>
<th>Source: relation type</th>
<th>Target: relation type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>☑ binary</td>
<td>☑ binary</td>
<td></td>
</tr>
<tr>
<td></td>
<td>☐ n-ary</td>
<td>☑ n-ary</td>
<td></td>
</tr>
</tbody>
</table>

A binary relation type references a source and a target object type. The source and target object type of the source relation type must be mapped to the source and target of the target relation type.

<table>
<thead>
<tr>
<th>ID: R2/1/2</th>
<th>Feature: arity</th>
<th>Source: relation type</th>
<th>Target: relation type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>☐ binary</td>
<td>☐ binary</td>
<td></td>
</tr>
<tr>
<td></td>
<td>☑ n-ary</td>
<td>☑ n-ary</td>
<td></td>
</tr>
</tbody>
</table>

An n-ary relation type references an infinite number of object type. References of a source relation type must be mapped to references of a target relation type. Often, a reference between a relation type and an object type is identified by an unique identifier (e.g. name). This identifier must be considered during the mapping. Furthermore, object types of the relation type in the source meta-model must be set to the relation type in the target meta-model.

<table>
<thead>
<tr>
<th>ID: R2/1/3</th>
<th>Feature: arity</th>
<th>Source: relation type</th>
<th>Target: relation type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>☑ binary</td>
<td>☐ binary</td>
<td></td>
</tr>
<tr>
<td></td>
<td>☑ n-ary</td>
<td>☑ n-ary</td>
<td></td>
</tr>
</tbody>
</table>

A binary relation type has two object types and an n-ary relationship type can have two or more references to object types. Thus, the mapping of a binary relation type to an n-ary relation type is trivial. Both references of the source relation type must be transformed into references of the n-ary relation type. Often the references of a binary relation type have a pre-defined name (e.g. start, end, source or target). The reference names of an n-ary relation type are not pre-defined. Therefore, the mapping can require the creation of names for the references of the created n-ary relation type.

Table 5.7: Mapping of relation types: arity – continued on next page
There are different options for the mapping of n-ary relation types. The mapping of an n-ary relation type with two references \((n = 2)\) to a binary relation type is no problem. In this case, both references of the n-ary must be mapped to the two references of the binary relation type. The mapping must consider the direction of the n-ary relation type.

The mapping of an n-ary relationship with more than two references \((n > 2)\) into a binary relation type requires a more complex mapping. We can differentiate between the following two solutions.

The first solution maps an n-ary relation type to a set of binary relation types. For each combination of two references of an n-ary relation type, a binary relation type must be created with two references to the corresponding object types. The number of binary relation types is equal with the Gaussian sum formula defined as \(\Delta_n = \frac{n(n+1)}{2}\). The problem of this mapping is that the correlation between the object types of the n-ary relation type is lost because the separation into binary relation types.

The second solution maps an n-ary relation type to an object type with additional binary relation types (see rule R3). This mapping strategy is also known as association class. In this case, the created object type in the target meta-model is interpreted as a relation type. We denote this created object type as object-relation type. Each reference of the n-ary relation must be transformed into a relation type in the target meta-model. The target of each created relation type is an object type which is equivalent to the object type of the source meta-model. The source of each created relation type is the previously created object-relation type. This mapping variant preserves the correlation between participating object types because the object type corresponding to the n-ary relation type joins all binary relation types. The number of created binary relation types is equal with the number \(n\) of references in the n-ary relation type. The disadvantage is the conversion of a relation type into an object type.

Table 5.7: Mapping of relation types: arity – continued from previous page
Multiplicity

<table>
<thead>
<tr>
<th>ID: R2/2/1</th>
<th>Source: relation type</th>
<th>Target: relation type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature: multiplicity</td>
<td>☒ fix (0..n)</td>
<td>☒ fix (0..n)</td>
</tr>
<tr>
<td>n, 0..1, 0..n, 1..n</td>
<td>n, 0..1, 0..n, 1..n</td>
<td></td>
</tr>
</tbody>
</table>

In this case, a mapping is unnecessary because source and target relation types use a fixed default multiplicity.

<table>
<thead>
<tr>
<th>ID: R2/2/2</th>
<th>Source: relation type</th>
<th>Target: relation type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature: multiplicity</td>
<td>☐ fix (0..n)</td>
<td>☒ fix (0..n)</td>
</tr>
<tr>
<td>n, 0..1, 0..n, 1..n</td>
<td>n, 0..1, 0..n, 1..n</td>
<td></td>
</tr>
</tbody>
</table>

A mapping of this feature configuration is trivial because the source multiplicity can be mapped to the target multiplicity without information loss.

<table>
<thead>
<tr>
<th>ID: R2/2/3</th>
<th>Source: relation type</th>
<th>Target: relation type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature: multiplicity</td>
<td>☔ fix (0..n)</td>
<td>☒ fix (0..n)</td>
</tr>
<tr>
<td>n, 0..1, 0..n, 1..n</td>
<td>n, 0..1, 0..n, 1..n</td>
<td></td>
</tr>
</tbody>
</table>

In this case, the fixed multiplicity of the source relation type must be mapped to the configurable multiplicity of the target relation type. Normally, there is no information loss.

<table>
<thead>
<tr>
<th>ID: R2/2/4</th>
<th>Source: relation type</th>
<th>Target: relation type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature: multiplicity</td>
<td>☐ fix (0..n)</td>
<td>☔ fix (0..n)</td>
</tr>
<tr>
<td>n, 0..1, 0..n, 1..n</td>
<td>n, 0..1, 0..n, 1..n</td>
<td></td>
</tr>
</tbody>
</table>

In this case, a mapping can lead to an information loss because the target multiplicity supports only one specific case of the source multiplicity. If the target multiplicity is equal or greater than the source multiplicity, then the transformation of model data leads to no element loss.

Table 5.8: Mapping of relation types: multiplicity

Composition

<table>
<thead>
<tr>
<th>ID: R2/3/1</th>
<th>Source: relation type</th>
<th>Target: relation type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature: composition</td>
<td>☒ support</td>
<td>☒ support</td>
</tr>
<tr>
<td>☐ no support</td>
<td>☐ no support</td>
<td></td>
</tr>
</tbody>
</table>

A mapping of this feature combination is trivial.

<table>
<thead>
<tr>
<th>ID: R2/3/2</th>
<th>Source: relation type</th>
<th>Target: relation type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature: composition</td>
<td>☐ support</td>
<td>☐ support</td>
</tr>
<tr>
<td>☔ no support</td>
<td>☔ no support</td>
<td></td>
</tr>
</tbody>
</table>

A mapping of this feature combination is trivial.

Table 5.9: Mapping of relation types: composition – continued on next page
5.4 Mapping of Meta-Modeling Concepts

<table>
<thead>
<tr>
<th>ID: R2/3/3</th>
<th>Feature: composition</th>
<th>Source: relation type</th>
<th>Target: relation type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>☐ support</td>
<td>☐ support</td>
<td>☐ support</td>
</tr>
<tr>
<td></td>
<td>☒ no support</td>
<td>☒ no support</td>
<td>☐ no support</td>
</tr>
</tbody>
</table>

A mapping of this feature combination is trivial.

<table>
<thead>
<tr>
<th>ID: R2/3/4</th>
<th>Feature: composition</th>
<th>Source: relation type</th>
<th>Target: relation type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>☒ support</td>
<td>☐ support</td>
<td>☐ no support</td>
</tr>
<tr>
<td></td>
<td>☐ no support</td>
<td>☒ no support</td>
<td>☒ no support</td>
</tr>
</tbody>
</table>

In this case, the composition feature is lost during the mapping. A solution for the missing composition could be a re-implementation of the composition feature as a trigger. The trigger deletes child elements after the deletion of a parent element.

Table 5.9: Mapping of relation types: composition – continued from previous page

**Object-Set**

<table>
<thead>
<tr>
<th>ID: R2/4/1</th>
<th>Feature: object-set</th>
<th>Source: relation type</th>
<th>Target: relation type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>☒ object</td>
<td>☒ object</td>
<td>☒ object</td>
</tr>
<tr>
<td></td>
<td>☐ object-set</td>
<td>☐ object-set</td>
<td>☐ object-set</td>
</tr>
</tbody>
</table>

The transformation of this feature combination is trivial.

<table>
<thead>
<tr>
<th>ID: R2/4/2</th>
<th>Feature: object-set</th>
<th>Source: relation type</th>
<th>Target: relation type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>☐ object</td>
<td>☒ object</td>
<td>☒ object</td>
</tr>
<tr>
<td></td>
<td>☒ object-set</td>
<td>☒ object-set</td>
<td>☒ object-set</td>
</tr>
</tbody>
</table>

The transformation of this feature combination is trivial.

<table>
<thead>
<tr>
<th>ID: R2/4/3</th>
<th>Feature: object-set</th>
<th>Source: relation type</th>
<th>Target: relation type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>☒ object</td>
<td>☐ object</td>
<td>☐ object-set</td>
</tr>
<tr>
<td></td>
<td>☐ object-set</td>
<td>☒ object-set</td>
<td>☒ object-set</td>
</tr>
</tbody>
</table>

This feature combination requires the creation of two object-sets in the target meta-model. The first object-set contains the source object type and the second object-set contains the target object type. There is no information loss.

Table 5.10: Mapping of relation types: object-set – continued on next page
The mapping from an object-set to a relation type with single object types is more complex. The object-set must be split into single object types. For each combination of object types in the source and target object-set, a relation type must be created with a source and target object type. The number of created relation types is the cross product of the source and target object-set. The name of the source relation type must be transformed into a name of the created relation types with an additional identification string (e.g. the name of the source and target object type). There is no information loss but the disadvantage of this mapping is the amount of relation types created.

Table 5.10: Mapping of relation types: object-set – continued from previous page

### Role type

<table>
<thead>
<tr>
<th>ID: R2/5/1</th>
<th>Source: relation type</th>
<th>Target: relation type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature: role</td>
<td>☒ no role</td>
<td>☒ no role</td>
</tr>
<tr>
<td></td>
<td>☐ role</td>
<td>☐ role</td>
</tr>
</tbody>
</table>

If both meta-modeling languages do not support role types, the mapping must only consider the correct mapping between the source and target object of a relation type.

<table>
<thead>
<tr>
<th>ID: R2/5/2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature: role</td>
</tr>
<tr>
<td>☐ no role</td>
</tr>
<tr>
<td>☒ role</td>
</tr>
</tbody>
</table>

In this case, roles of a source relation type must be mapped to roles of the target relation type. This mapping can include the mapping of the role names and the corresponding object types. Normally, there is no information loss.

<table>
<thead>
<tr>
<th>ID: R2/5/3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature: role</td>
</tr>
<tr>
<td>☒ no role</td>
</tr>
<tr>
<td>☐ role</td>
</tr>
</tbody>
</table>

This feature combination requires an artificial creation of role types in the target meta-model. For each object type which is part of a relation type in the source meta-model, a corresponding role type must be created as part of a target relation type. The target relation type and the object type must be added to the created role type. Normally, the creation of an artificial role type leads to no information loss.
5.4 Mapping of Meta-Modeling Concepts

In this case, only the source meta-modeling language supports a role concept but the target meta-modeling language does not offer a role concept. There are two mapping variants.

The first variant is the mapping of an object, role and relation type from a source meta-model into an object and relation type in a target meta-model. The created relation type and participating object type are connected directly without a role type in the target meta-model. The deletion of the role type can lead to information loss.

The second variant maps a role type to an object type. This created object type must be connected with a relation type and object type corresponding to the relation and object type from the source meta-model. This mapping leads can leads to a problem because the unclear differentiation between normal object types and object types created for a role type.

Table 5.11: Mapping of relation types: role – continued from previous page

5.4.3 Attribute Mapping

Analogous to the mapping of relation types, in this section we discuss the mapping of attributes. We assume that attributes are mapped to attributes (see rule R4 in Table 5.6), hence, we also focus on mappings between different feature combinations. The marked rectangles in the feature row show the current configuration of the source and target attribute.

**Multiplicity**

<table>
<thead>
<tr>
<th>ID: R4/1/1</th>
<th>Source: attribute</th>
<th>Target: attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature: multiplicity</td>
<td>☒ single-value</td>
<td>☒ single-value</td>
</tr>
<tr>
<td></td>
<td>☐ multi-value</td>
<td>☐ multi-value</td>
</tr>
</tbody>
</table>

This combination is trivial and leads to no information loss.

<table>
<thead>
<tr>
<th>ID: R4/1/2</th>
<th>Source: attribute</th>
<th>Target: attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature: multiplicity</td>
<td>☐ single-value</td>
<td>☐ single-value</td>
</tr>
<tr>
<td></td>
<td>☒ multi-value</td>
<td>☒ multi-value</td>
</tr>
</tbody>
</table>

This combination is also trivial and leads to no information loss.

Table 5.12: Mapping of attributes: multiplicity – continued on next page
In this case, a single-value must be represented as one element in a multi-value.

Normally, this feature combination leads to an information loss because a single value can only hold one value. Nevertheless, a possible mapping of this feature combination requires the encoding of multi-values as single-values. For instance, a multi-value of a source attribute must be concatenated to a single-value of a target attribute. The concatenation uses a conversion of a multi-value into a string which contains every entry as one value separated by a delimiter character. This single-value can be stored in the target attribute. The reading of this value requires the separation and selection of values. Problems can be, for instance, the handling of the delimiter character or the limited size of a single-value.

Table 5.12: Mapping of attributes: multiplicity – continued from previous page

Ordered

Table 5.13: Mapping of attributes: ordered – continued on next page
5.4 Mapping of Meta-Modeling Concepts

<table>
<thead>
<tr>
<th>ID: R4/2/4</th>
<th>Feature: ordered</th>
<th>Support</th>
<th>No Support</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>✓ support</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>✓ no support</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>□ support</td>
<td></td>
<td>✓ no support</td>
</tr>
<tr>
<td></td>
<td>□ no support</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Generally there is no problem to transform an unordered attribute into an ordered attribute.

Table 5.13: Mapping of attributes: ordered – continued from previous page

**Unique**

<table>
<thead>
<tr>
<th>ID: R4/3/1</th>
<th>Source: attribute</th>
<th>Target: attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>✓ support</td>
<td>✓ support</td>
</tr>
<tr>
<td></td>
<td>✓ no support</td>
<td>✓ no support</td>
</tr>
<tr>
<td></td>
<td>□ support</td>
<td>□ no support</td>
</tr>
<tr>
<td></td>
<td>□ no support</td>
<td>□ no support</td>
</tr>
</tbody>
</table>

This feature combination is trivial and leads to no information loss.

<table>
<thead>
<tr>
<th>ID: R4/3/2</th>
<th>Feature: unique</th>
<th>Support</th>
<th>No Support</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>✓ support</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>✓ no support</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>□ support</td>
<td></td>
<td>✓ no support</td>
</tr>
<tr>
<td></td>
<td>□ no support</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This feature combination is also trivial and leads to no information loss.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>✓ support</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>✓ no support</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>□ support</td>
<td></td>
<td>✓ no support</td>
</tr>
<tr>
<td></td>
<td>□ no support</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In this case, the information about the unique feature is lost. The values can be transformed from a source to a target attribute but the uniqueness cannot guaranteed by the target attribute. Maybe additional constraints on the attribute can re-implement the uniqueness of values.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>✓ support</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>✓ no support</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>□ support</td>
<td></td>
<td>✓ no support</td>
</tr>
<tr>
<td></td>
<td>□ no support</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

There is no need to map a non-unique attribute to a unique attribute. Nevertheless, in the case of a mapping, a transformation must ensure that the source values are unique. Otherwise, this transformation can lead to an error during the transformation.

Table 5.14: Mapping of attributes: unique
5.4.4 Mapping of Inheritance

<table>
<thead>
<tr>
<th>ID: R8/1</th>
<th>Source: concept</th>
<th>Target: concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature:</td>
<td>☒ no support</td>
<td>☒ no support</td>
</tr>
<tr>
<td>inheritance</td>
<td>☐ single-inheritance</td>
<td>☐ single-inheritance</td>
</tr>
<tr>
<td></td>
<td>☐ multi-inheritance</td>
<td>☐ multi-inheritance</td>
</tr>
</tbody>
</table>

This mapping is no problem.

<table>
<thead>
<tr>
<th>ID: R8/2</th>
<th>Source: concept</th>
<th>Target: concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature:</td>
<td>☐ no support</td>
<td>☒ no support</td>
</tr>
<tr>
<td>inheritance</td>
<td>☒ single-inheritance</td>
<td>☐ single-inheritance</td>
</tr>
<tr>
<td></td>
<td>☒ multi-inheritance</td>
<td>☐ multi-inheritance</td>
</tr>
</tbody>
</table>

In this case, a single-inheritance in a source meta-model must be expressed without inheritance in a target meta-model. Depending on the inheritance implementation, one possible strategy is the transformation of all concepts in the inheritance tree from the source meta-model into the target meta-model. During this transformation, the combination of all properties in an inheritance tree path must be assigned to the created concept in the target meta-model. Properties can include, for instance, attributes or relationships.

<table>
<thead>
<tr>
<th>ID: R8/3</th>
<th>Source: concept</th>
<th>Target: concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature:</td>
<td>☐ no support</td>
<td>☒ no support</td>
</tr>
<tr>
<td>inheritance</td>
<td>☒ single-inheritance</td>
<td>☐ single-inheritance</td>
</tr>
<tr>
<td></td>
<td>☒ multi-inheritance</td>
<td>☐ multi-inheritance</td>
</tr>
</tbody>
</table>

In this case, single-inheritance can be expressed by multi-inheritance without information loss.

<table>
<thead>
<tr>
<th>ID: R8/4</th>
<th>Source: concept</th>
<th>Target: concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature:</td>
<td>☐ no support</td>
<td>☒ no support</td>
</tr>
<tr>
<td>inheritance</td>
<td>☐ single-inheritance</td>
<td>☒ single-inheritance</td>
</tr>
<tr>
<td></td>
<td>☒ multi-inheritance</td>
<td>☐ multi-inheritance</td>
</tr>
</tbody>
</table>

The mapping from multiple-inheritance to single-inheritance is possible but not reasonable. The mapping requires a re-structuring of the inheritance tree. For instance, two super-elements of a sub-element in the source meta-model can be combined into one super-element in the target meta-model. But the combination must consider a merge of properties.

Table 5.15: Mapping of inheritance

5.4.5 Further Transformation Aspects

Mapping Strategy

The mapping of meta-modeling languages is a complex problem. Often, there are different options for the definition of mappings. For this reason, we discuss some aspects which should be taken into consideration during the development of a mapping. The first aspect is the completeness of a mapping. The mapping
of concepts should be complete as much as possible, that means, each concept in the source meta-modeling language should be mapped to concepts in the target meta-modeling language with minimal information loss.

One further aspect is the usability or practicality of mappings. If there is no simple one-to-one mapping, concepts in the source meta-model must be expressed with a combination of concepts in the target meta-model. An example of this issue is the mapping of an object-set relation type into a binary object-relation type (see rule R2/4/4). In this case, the transformation can lead to a huge number of relation types in the target meta-model. Regarding usability, the decrease of completeness can, in some cases, be reasonable in order to increase the usability of mappings and created target meta-models.

A third aspect concerns the connection between the M2- and M1-transformation. Rules for the M2-transformation influence the realization of the M1-transformation and vice versa. Complex rules at the M2-level lead to complex rules at the M1-level. Furthermore, the transformation rules at the M1-level influence the mapping possibilities at the M2-level. Hence, a mapping should strike a balance between completeness and complexity.

In summary, the aspects completeness, usability, dependency between M2- and M1-transformation, and complexity must be taken into account when defining a mapping.

Structuring and Dependency

There are different approaches to structure modeling languages (see Table 5.5). One concept is the grouping of modeling concepts into logical units. There is no general mapping rule for these concepts but we recommend to map these grouping concepts because these concepts influence, for instance, the identification of metamodel elements. Additional to these grouping concepts, meta-models have a defined structure with dependencies between elements. These dependencies must be taken into account during the mapping. Similar to the grouping concept, there is no general mapping rule because the high variety of dependencies between elements.

Constraint Languages

There are different mechanisms for the definition of modeling constraints (see Section 5.3.2). The transformation between these different constraint languages is a challenging task and difficult to realize. We do not focus on this problem but we want to point out that this aspect must be taken into account during a mapping.

Identifiers

Transformation of meta-models requires transformation of element identifiers. The specification of identifiers depends often on the concrete meta-modeling environment. There can be a difference in valid names or in the usage of internal and external identifiers. The right transformation of identifiers is important for the correct creation of concepts in target meta-models. Furthermore, identifiers are necessary
to create relations between participating meta-model elements. These relations (transformation instance) are required for the execution of the M1-transformation.

**Datatype**

Datatypes defines the range of values and possible operation on these values. Many meta-modeling environments support common datatypes, such as string, double, integer, and date, but many environments use their own definition which leads to slightly different datatypes. These differences require the conversion of data types during the transformation.

**5.5 Summary**

In this chapter, we analyzed different meta-modeling languages and extracted typical meta-modeling concepts with their features. In general, all investigated meta-metamodels follow an object-oriented approach and support at least the following basic concepts: object type, relation type, and attribute. Additionally, some meta-metamodels support the concept of model type. There are strong differences in the support of relation types. The simplest realization of a relation type is the usage of a reference. In contrast to this, the usage of an n-ary object-set relation is a more powerful and complex realization of a relation type.

The second part of this chapter described mapping rules between the previously extracted meta-modeling concepts. We presented mappings between basic concepts and between different features of these concepts. We discussed some further mapping aspects which must be taken into account during the mapping of meta-modeling languages. The mapping rules are suggestions and should be adaptable to other meta-modeling languages conforming to the concept framework. We will apply the mapping rules in next section.
6 Application of M3-Level-Based Bridges

This chapter presents three bridges between different meta-modeling environments. These bridges should demonstrate the general applicability of the M3B approach. In the first part, we describe the ARIS-EMF, MetaEdit-EMF and Visio-EMF bridge. In the second part, we present the combination of the MetaEdit-EMF and Visio-EMF bridge.

6.1 Overview

The description of each bridge follows the same structure. Firstly, we give an overview of the bridge including the motivation for building a bridge, the input and output of a bridge, and supported bridge features. Afterwards, we define mapping rules for the transformation of meta-models and models. The foundation for the development of these rules is based on the mapping rules suggested in Chapter 5.4. We select suitable rules and adapt these rules to the concrete meta-metamodels. Additionally to a textual description of the mapping rules, we provide a more formal description in ETL (see Section 2.2.4). The ETL rules are presented in Appendix B. After the mapping description, we explain the bridge implementation and present a demonstration of the bridge. Each bridge realizes a small use case which enables the migration of Event-Driven Process Chain models.

After describing the bridges, we present the combination of the Visio-EMF and MetaEdit-EMF bridge. The combination uses EMF as a central element for the connection because both bridges include EMF as meta-modeling environment. The combination allows the exchange of models between Visio and MetaEdit+.

6.2 Bridge 1: ARIS and Eclipse Modeling Framework

6.2.1 Bridge Architecture

ARIS is a modeling environment suitable for business process modeling in large companies. In contrast to ARIS, EMF is a universal technology suitable for generic model processing, such as transformation, validation or code generation. The motivation for implementing the bridge is based on the idea to extend the model processing capabilities of ARIS by using EMF technologies. The bridge allows the migration of models from ARIS to EMF [Kern 2007b; Kern and Kühne 2007a; Kern 2008b]. These migrated EMF models can be processed with EMF tools (e.g. Eclipse Epsilon).
Figure 6.1 illustrates an overview of the bridge. The ARIS-EMF bridge consists of two transformations. The M2-transformation enables the creation of EMF metamodels. The input of this M2-transformation is an ARIS filter. A filter is a subset of the complete ARIS method and includes a selection of modeling elements. Both the ARIS method and filter conform to the ARIS meta-metamodel. The M2-transformation supports only one direction from ARIS to EMF because ARIS does not allow the automatic creation of ARIS filters. The created EMF meta-model extends an abstract ARIS meta-model. This abstract meta-model covers two tasks. The first task is the emulation of the storage structure for ARIS models. The second task concerns the differentiation of meta-modeling concepts. ARIS supports more concepts than EMF. To distinguish the ARIS concepts in the EMF meta-model, we use different super classes in the abstract ARIS meta-model. The abstract ARIS meta-model and the imported ARIS-EMF filter form the complete meta-model in EMF.

The M1-transformation is responsible for the migration of models from ARIS to EMF. The M1-transformation is directed from ARIS to EMF. The created EMF models conform to the EMF meta-model previously created by the M2-transformation. The reverse direction (from EMF to ARIS) is not realized.

![Diagram](image-url)

Figure 6.1: Bridge between ARIS and EMF
6.2.2 M2-Transformation

The M2-transformation between ARIS and EMF is responsible for the creation of EMF meta-models and includes the following rules. These transformation rules are also described as pseudo code in Appendix B.1, Listing B.1.

**Filter** \(\rightarrow\) **EPackage**  
Starting point of the transformation is an ARIS filter. A filter defines a container for other meta-model elements. The corresponding concept in Ecore is an EPackage. An EPackage can also contain meta-model elements, more precisely EClasses. This rule creates a corresponding EPackage for each filter. The name of an EPackage is equal to the name of a filter. This rule executes further rules for the transformation of object, connection, and model types.

**Object Type** \(\rightarrow\) **EClass**  
In ARIS, an object type represents a set of model entities. Object types are mapped to EClasses. In this case, we apply rule R1 from the rules suggested in Table 5.6. The name of an object type is equal to the name of the created EClass. Object types can have attributes. Thus, this rule calls a rule for the mapping of attributes.

**Connection Type** \(\rightarrow\) **EClass**  
A connection type in ARIS is a binary object-set relation type between two sets of object types. We apply rule R3 from Table 5.6. There are two possibilities for the transformation. The first possibility is a mapping of connection types to EReferences. This option is a simple solution with the problem that a connection type can have attributes and EReferences cannot. Thus, the usage of EReference would lead to a loss of attributes. The alternative option is using EClass, which can have attributes. Therefore, each connection type is transformed into an EClass. The name of a connection type is assigned to the name of the created EClass. One further problem is that a native EClass does not represent a relation type because an EClass has no connections to other elements. For this reason, a binary relation is emulated by an EClass with two additional EReferences, but these special EReferences are on the same language level as normal attributes. This mixture may lead to a conflict. Thus, normal attributes and both of these references must have different names to avoid a conflict. Besides the attribution, the object-set feature of connection types is a further issue. The creation of the cross product for each source and target object type would lead to a huge amount of EClasses and EReferences. For practical reasons, we use only one EClass with a source and target reference. The target of these references is the general EClass with the name Object Definition. This solution occurs an information loss but eases the handling of references.

**Model Type** \(\rightarrow\) **EPackage and EClass**  
A model type in ARIS defines a set of similar models. A model type comprises a set of symbols and possible connections between these symbols. In Ecore, there is no native model type concept. The only way for a mapping is a transformation of each model type into an EClass. In this case, we apply rule R6 from the rules suggested in Table 5.6. The name of the
created EClass is equal to the model type name. A model type can have attributes.
The transformed attributes are part of the created EClass. Furthermore, a model
type in ARIS has a namespace character for its containing symbols. Therefore, we
create an EPackage for each model type. This EPackage contains the EClass which
corresponds to the model type. Furthermore, this rule calls the mapping rules for
symbols.

**Symbol $\mapsto$ EClass**  A symbol defines a set of graphical modeling elements rep-
represented as icons in a model. Each symbol is transformed into an EClass. The
EClass name is equal to the symbol name. A symbol has no further properties on
the language level.

**Attribute $\mapsto$ EAttribute**  Both meta-modeling languages allow the definition
of attributes. We apply rule R4 from Table 5.6 and transform ARIS attributes
into EAttributes. The EAttribute name is equal to the ARIS attribute name. The
EAttribute datatype is equivalent to the ARIS attribute datatype. If there is an
equivalent datatype in EMF, then we use this datatype, for instance, date to date
or integer to integer. If there is no equivalent datatype, we map this datatype to
the string datatype. This rule is used during the transformation of object types,
connection types, and model types.

**Abstract ARIS meta-model in EMF**  The M2-transformation reads an ARIS
filter and creates a corresponding meta-model in EMF. This meta-model rep-
resents language concepts of the transformed ARIS filter. But the transformation of
models into EMF needs further structure elements. Therefore, we use a abstract
meta-model structure for storing imported ARIS models. This generic meta-model
forms the connection between meta-model concepts (language aspect) and generic
model elements. The abstract meta-model is illustrated in Figure 6.2. The meta-
model consists of groups. A group contains subgroups and further model elements
(e.g. object definitions, connection definitions and models). Models contain object
occurrences and connection occurrences. The abstract meta-model is connected
to the created EMF meta-model by inheritance relationships. Thus, the abstract
meta-model defines the complete space for ARIS models and the created meta-
model restricts and refines this space, respectively.

### 6.2.3 M1-Transformation

The M1-transformation migrates models from ARIS into EMF. The transformation
at the M1-level includes the following rules. These rules are also described as pseudo
code in Appendix B.1, Listing B.2.

**Group**  The M1-transformation starts with the import of ARIS groups. For each
ARIS group a corresponding group in EMF is created with an identical group name
and child groups. The transformation of child groups requires an iteration over all
6.2 Bridge 1: ARIS and Eclipse Modeling Framework

Figure 6.2: Abstract ARIS meta-model in EMF

groups and child groups. This rule calls further rules for the transformation of object definitions, connection definitions and models.

Object Definition In contrast to the group rule, this rule depends on a type mapping and corresponds to the rule $\text{ObjectType} \mapsto \text{EClass}$. The rule iterates over each object definition in ARIS and queries the object type. Afterwards, the rule selects a corresponding EClass and creates an EObject. All attribute values are copied to the attributes of this EObject. The created EObject is assigned to a group.

Connection Definition This rule creates an EObject for each connection definition. This rule depends on the rule $\text{ConnectionType} \mapsto \text{EClass}$ from the metamodel mapping. That is, the rule queries the EClass corresponding to a connection type and creates an EObject as an instance of this EClass. After the creation of the EObject, the rule migrates attribute values and sets the source and target object definition. The created EObject is assigned to a group.

Model This rule correlates to the rule $\text{ModelType} \mapsto \text{EClass}$ from the metamodel transformation and creates in EMF an EObject for each ARIS model. For this reason, the rule looks for the EClass corresponding to the ARIS model type and instantiates this EClass. The created EObject is assigned to a group. Furthermore, attribute values from the ARIS model are assigned to the EAttributes of the EObject.

Object Occurrence This rule refers to the rule $\text{Symbol} \mapsto \text{EClass}$ of the metamodel transformation. The rule transforms an ARIS object occurrence into an EObject. This EObject is an instance of the EClass correlating to the type of the current object occurrence. In this case, the type of an object occurrence is a symbol. Additionally, the rule sets the previously transformed ARIS object definition and assigns the EObject to a model EObject.
Connection Occurrence  This rule is independent of the meta-model transformation. The rule instantiates the EClass with the name *Connection Occurrence* for each connection occurrence in ARIS. The rule sets the source and target object occurrences as source and target of the created EObject. Furthermore, the created EObject is assigned to a model EObject.

6.2.4 Implementation

The ARIS-EMF bridge is implemented in ARIS script. ARIS script offers an API which enables the processing of model data. The bridge comprises two scripts. The first script is responsible for the meta-model transformation from ARIS to EMF. The script reads a selected filter and produces an Ecore meta-model. The script implements the rules from Section 6.2.2. We use the native EMF library for the creation of Ecore elements. This is possible because ARIS script allows the integration of Java libraries. Additionally to the implementation of the transformation rules, the script implements a set of dialogs allowing the selection of a filter as source and a file location of the Ecore model as target.

The second script is responsible for the model transformation from ARIS to EMF. The script reads a selected group with its elements and produces an EMF model. The script implements the rules from Section 6.2.3. The model transformation needs the mapping between meta-model elements. Therefore, the model transformation first executes the meta-model transformation. The produced type mapping (transformation instance) is stored in a hash map. The model transformation iterates over model elements, queries the source type, looks up the target type in the type mapping and instantiates the target type in order to create an element in the target model. After the transformation process, the EMF model is serialized as an XMI file. Additionally to the implementation of the transformation rules, the script implements a set of dialogs in order to select an ARIS group as source and a file location of the EMF model as target.

6.2.5 Example

In this section, we provide an example of the bridge. Figure 6.3 shows the migration of EPC models. The ARIS filter in the left top corner (Figure 6.3(a)) consists of elements, such as *Event*, *Function*, activates, creates, and *EPC*. The meta-model is shown as a table because ARIS offers no graphical visualization for their filters. The M2-transformation creates the corresponding Ecore meta-model in the right top corner (Figure 6.3(b)). The screenshot in the left bottom corner (Figure 6.3(c)) shows an EPC model in ARIS. The M1-transformation creates the EMF model in the right bottom corner (Figure 6.3(d)). This model is represented as a tree because EMF has no graphical representation.
6.2 Bridge 1: ARIS and Eclipse Modeling Framework

<table>
<thead>
<tr>
<th>Object type</th>
<th>Event (OT_EVT)</th>
<th>Function (OT_FUNC)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Connection type</th>
<th>Activates (CT_ACTIV_1)</th>
<th>Creates (CT_CRT_1)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Model type</th>
<th>EPC (MT_EEPC)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Event (ST_EV)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Function (ST_FUNC)</th>
</tr>
</thead>
</table>

(a) EPC meta-model in ARIS

(b) EPC meta-model in EMF

(c) EPC model in ARIS

(d) EPC model in EMF

Figure 6.3: Transformation example of ARIS-EMF bridge
6.3 Bridge 2: MetaEdit+ and Eclipse Modeling Framework

6.3.1 Bridge Architecture

The MetaEdit-EMF bridge is the second application of the M3B approach. The bridge allows the migration of models from MetaEdit+ to EMF and vice versa [Kern 2008a]. MetaEdit+ is a tool for Domain-Specific Modeling. The tool allows the creation of self-defined languages by using a meta-modeling approach. The motivation for building this bridge is similar to the ARIS-EMF bridge. The bridge allows the combination of both technologies to benefit from each other.

The overview of the bridge is shown in Figure 6.4. The mapping at the M3-level describes the conceptual relation between GOPPRR (MetaEdit+) and Ecore (EMF). The M2-transformation implements this mapping and enables the transformation of GOPPRR meta-models into EMF meta-models. The transformation is unidirectional from MetaEdit+ to EMF. The created EMF meta-model extends an abstract MetaEdit+ meta-model. This abstract meta-model and the imported GOPPRR meta-model form a complete meta-model in EMF. Similar to the ARIS-EMF bridge, the abstract MetaEdit+ meta-model has two tasks. The first task is the storage emulation for MetaEdit+ models. The second task concerns different
meta-modeling concepts. GOPPRR support more concepts than EMF. Different super classes are necessary to distinguish these concepts in EMF.

The M1-transformation allows the conversion of models from MetaEdit+ to EMF and vice versa. In contrast to the meta-model transformation, the model transformation allows both directions. The direction from EMF to MetaEdit+ is only possible if an EMF model conforms to an EMF meta-model previously created by the M2-transformation.

6.3.2 M2-Transformation

The meta-model transformation maps meta-models from MetaEdit+ to EMF. Starting point of the transformation is a GOPPRR meta-model contained in a project. The transformation creates a named EPackage for each GOPPRR meta-model. The transformation uses the following rules. These rules are also described as pseudo code in Appendix B.2, Listing B.3.

Graph Type $\rightarrow$ EClass A graph type comprises object types, relationship types, and role types. A graph type is equal to a model type and defines a set of graphs or models, respectively. Instances of a graph type are represented as diagrams. Ecore does not support a model type concept. However, we suggest to transform each graph type into an EClass. In this case, we apply rule R6 from the suggested rules in Table 5.6. The name of a graph type is equal to the name of a created EClass. The inheritance between graph types is mapped to the inheritance between the created EClasses.

Object Type $\rightarrow$ EClass An object type in MetaEdit+ defines a set of model entities. These model entities or instances are represented by a symbol in a diagram. The corresponding concept in Ecore is the EClass concept which can also define a set of model entities. Therefore, this rule maps an object type into an EClass (see rule R1 in Table 5.6). The type name of an object type is equal to the EClass name. The inheritance between object types is mapped to the inheritance between EClasses.

Relationship Type $\rightarrow$ EClass A relationship type defines a set of model entities serving as a connection between different objects. An instance of a relationship type is represented as a line between objects. There are two possibilities for a mapping. The first option is a mapping as reference. This option leads to a loss of attributes and inheritance between relationship types because an EReference does not support attributes and inheritance. A further issue is that GOPPRR supports n-ary relations but Ecore only supports binary relations. Hence, the second option is a mapping of a relationship type into an EClass. In this case, we apply rule R2/1/4 from the suggested rules in Table 5.7. This mapping option allows the transformation of n-ary relations, attributes and inheritance. The type name of the relationship type is equal to the EClass name. The properties of a relationship type are transformed by another rule.
Role Type $\rightarrow$ EClass A relationship type in GOPPRR references role types. A role type defines a set of roles. A role describes how objects take part in a relationship. Analogous to the relationship type, there are two options for the role mapping. The first option is the usage of EReferences. In this case, attributes and inheritance cannot be transformed to EMF. The second and preferred option is the usage of EClass. In this case, we apply rule R2/5/4 from the suggested rules in Table 5.11. Each role type is transformed to an EClass with references to the previously created EClass corresponding to the GOPPRR relationship type. The name of the role type is equal to the one of the created EClass. The properties of a role type and inheritance between role types are mapped to the created EClass.

Property $\rightarrow$ EAttribute or EReference Properties in MetaEdit+ can hold values or can reference other model elements (i.e. objects or graphs). In Ecore, there are two equivalent concepts: EAttribute and EReference. This mapping rule is divided into two parts. (1) If a property holds values, then this property is transformed into an EAttribute. The datatype of the property is mapped to an equivalent datatype in Ecore. (2) If a property references other model elements, then this property is transformed into an EReference. The datatype of the GOPPRR property is another element in the GOPPRR meta-model (e.g. object types or graph types). Hence, the created EReference points to the EClass corresponding to the type of the GOPPRR property element. Furthermore, the name of the created EAttribute or EReference is equal to the GOPPRR property name. MetaEdit+ and Ecore support a cardinality. This rule also maps the cardinality value to the created EReference or EAttribute.

Abstract MetaEdit+ meta-model in EMF The M2-transformation reads a GOPPRR meta-model and creates a corresponding meta-model in EMF. The created EMF meta-model represents language concepts of the GOPPRR meta-model. But the M1-transformation needs further structure elements. Therefore, we use an abstract meta-model for storing imported GOPPRR models. This generic meta-model is the connection between meta-model concepts (language aspect) and generic model elements. The abstract meta-model is illustrated in Figure 6.5. The abstract meta-model comprises a project containing graphs. A graph contains further model elements: objects, relations, roles, and diagrams. Diagrams contain symbols which have a reference to an object. The abstract meta-model is connected to the previously created EMF meta-model by inheritance. Thus, the abstract meta-model defines the complete space for GOPPRR models and the created meta-model restricts and refines this space, respectively.

6.3.3 M1-Transformation

The M1-transformation converts models from MetaEdit+ to EMF and back again. The transformation comprises the following rules. These rules are also described as pseudo code in Appendix B.2, Listing B.4.
Project  Starting point of the M1-transformation is a project in MetaEdit+. This rule creates in EMF an EObject of the EClass with the name Project for each project in MetaEdit+. A project contains further model elements (e.g. graphs, objects, ports, etc.). This rule iterates over these model elements and applies further transformation rules.

Graph  This rule depends on the rule GraphType ↦ EClass. The rule creates an EObject for each graph object. This EObject is an instance of the EClass relating to the graph type of a graph object. Furthermore, the rule sets different reference between the created EObject and further EObjects which represent other model elements (objects, relations). The rule iterates over properties of the graph object and sets values to the corresponding attributes of the created EObject.

Object  This rule depends on the rule ObjectType ↦ EClass. The rule converts each MetaEdit+ object into an EObject. The EClass of this created EObject correlates to the object type of the MetaEdit+ object. After the object creation, the rule copies property values to EAttribute or EReference values.

Relationship  This rule transforms relationships from MetaEdit+ to EMF. The rule depends on a type mapping and corresponds to the rule RelationshipType ↦ EClass. The rule iterates over all relationships and queries the type. Afterwards, the rule queries the EClass corresponding to this type and creates an instance of this EClass. After the EClass instantiation, the rule copies values and references from the MetaEdit+ relationship to the created EObject.

Role  This rule transforms role elements and depends on the rule RoleType ↦ EClass. The rule iterates over all role elements and selects the role type. Based
on the role type, the rule selects the corresponding EClass and creates an EObject of this EClass. The rule converts all property values and references to EAttribute or EReference values.

**Diagram and Symbol** This rule creates an instance of the EClass with the name *Diagram* for each MetaEdit+ diagram. Furthermore, for each symbol in the diagram an EObject of the EClass *Symbol* is created and assigned to the previously created diagram EObject. The position of the MetaEdit+ symbol and the relation to the MetaEdit+ object is copied to the symbol EObject. In MetaEdit+, diagrams and symbols have no type in a meta-model. Hence, the creation of diagrams and symbols is independent of the M2-transformation.

### 6.3.4 Implementation

The bridge is implemented in Java as an Eclipse plug-in and consists of three transformations and widgets to specify input parameters. The current version 1.1.0 of the bridge is available for download.\(^1\) There is a transformation for meta-models from MetaEdit+ to EMF, a transformation for models from MetaEdit+ to EMF, and a transformation offering the re-import of EMF models into MetaEdit+.

MetaEdit+ offers an XML export for meta-models. The XML schema is vendor-specific but well documented. The bridge implements a GOPPRR-XML reader which can read these XML files. Additionally, the reader enables the selection of meta-model elements, for instance, graph types, object types, or relationship types. The M2-transformation uses the GOPPRR-XML reader and applies the transformation rules from Section 6.3.2. The transformation creates EPackages, EClasses and other elements as in-memory objects and serializes these objects as XMI-Ecore file. The transformation uses the EMF Java API for the creation and serialization of EMF meta-models.

The M1-transformation implements the rules from Section 6.3.3. The transformation uses the SOAP API from MetaEdit+ for reading and writing model elements. The transformation navigates through MetaEdit+ models and creates EMF models as in-memory objects. The reverse transformation from EMF to MetaEdit+ reads EMF models with the Java API and uses the SOAP API to create models in MetaEdit+.

### 6.3.5 Example

In this section, we demonstrate the application of the bridge. This example uses the same EPC example as the demonstration of the ARIS-EMF bridge. The bridge transforms the EPC meta-model from MetaEdit+ into EMF. Afterwards, the bridge migrates an EPC model from MetaEdit+ into EMF.

Figure 6.6(a) illustrates an EPC meta-model in MetaEdit+. MetaEdit+ uses dialogs for the definition of languages. In addition to this, MetaEdit+ provides a visualization of language definitions. The EPC language definition consists of

\(^1\)http://www.informatik.uni-leipzig.de/~kern/metaedit.emf.bridge_1.1.0.jar
the following object types: Node, Event, Function, Connector, AND, XOR and OR. The gray arrows are inheritance relations between object types. The orange rhombus represents the relationship type Arc and the green circles are the role types: From and To. The M2-transformation creates the meta-model in Ecore at the right top corner (Figure 6.6(b)). The M1-level transformation reads the EPC model in the left bottom corner (Figure 6.6(c)) and generates the EMF model in the right bottom corner (Figure 6.6(d)).

Figure 6.6: Transformation example of MetaEdit-EMF bridge
6.4 Bridge 3: Microsoft Visio und Eclipse Modeling Framework

6.4.1 Bridge Architecture

The Visio-EMF bridge is the third application of the M3B approach. The bridge allows the migration of models from Visio to EMF and vice versa [Kern and Kühne 2009; Kern et al. 2009b]. The motivation for building this bridge is similar to the other two bridges. The bridge combines the strengths of both technologies. For instance, Visio is a powerful modeling tool and EMF offers good model processing tools. The combination allows a processing of Visio models with EMF tools.

An overview of the bridge is illustrated in Figure 6.7. The bridge consists of an M2-transformation enabling the conversion of Visio stencils into EMF meta-models. The transformation is unidirectional. The created EMF meta-model extends an abstract Visio meta-model. This abstract meta-model and the imported Visio meta-model forms the complete meta-model in EMF. Similar to the ARIS-EMF and MetaEdit-EMF bridge, the abstract meta-model covers two tasks. The first task is the definition of a general structure for models independent of their language definition. The second task concerns meta-modeling concepts. There are rules

---

**Figure 6.7: Bridge between Visio and EMF**
mapping different Visio concepts to the same Ecore concept. We use different super
classes in the abstract Visio meta-model to distinguish these concept in EMF.
The M1-transformation migrates models from Visio to EMF. The transformation
back from EMF to Visio is also possible under the condition that the EMF models
conform to a meta-model previously created by the M2-transformation.

6.4.2 M2-Transformation

The meta-model transformation maps a Visio stencil into an EMF meta-model.
The transformation applies the following rules. These rules are also described as
pseudo code in Appendix B.3, Listing B.5.

Stencil $\mapsto$ EPackage  A stencil contains a set of modeling concepts which are
denoted in Visio as masters. A similar concept in Ecore is EPackage also comprising
a set of modeling concepts in form of EClasses. Therefore, a stencil is mapped into
an EPackage. The name of the created EPackage is equal with to stencil name.

Object Master $\mapsto$ EClass  There are two types of masters: object master and
connection masters. An object master is similar to an object type and defines a
set of objects. In this case, we apply rule R1 from the suggested rules in Table 5.6.
The rule transforms each object master into an EClass. The EClass name is equal
to the master name. The EPackage of the created EClass corresponds to the stencil
of the master.

Connection Master $\mapsto$ EClass  The second type of masters is the connection
master. This kind of master is similar to a relation type. Instances are represented
as graphical lines between model elements. There are two mapping possibilities.
The first possibility is a mapping of a connection master into an EReference. The
problem of this mapping is the missing attributability of EReferences. But Visio
allows the attribution of connection masters. This mapping variant would lead to
a loss of attributes. The second and preferred option is the transformation of a
connection master into an EClass. In this case, we apply rule R3 from the suggested
rules in Table 5.6. This solution enables the mapping of attributes. The created
EClass is derived from a super class which has two references. These two references
implement the source and target of a relation. The master name is mapped onto
the EClass name. The EPackage of the created EClass is equal with the stencil of
the corresponding master.

Property $\mapsto$ EAttribute  This rule transforms each Visio property into an EAt-
tribute. A property of a master (object and connection) relates to the EAttribute
of the corresponding EClass. The names of the property and EAttribute must be
equal. The rule maps equivalent datatypes between EAttribute and property, for
instance, string to string or date to date.
Abstract Visio meta-model in EMF  The M2-transformation reads a stencil and creates an EMF meta-model. The created meta-model represents language concepts of the transformed Visio stencil. The M1-transformation of models needs further structure elements. Therefore, we use an abstract meta-model. This abstract meta-model is a connection between meta-model concepts (language aspect) and generic model elements. The abstract meta-model is illustrated in Figure 5.9 (see Section 5.2.7). The meta-model consists of documents, a document includes pages, and a page contains shapes. These shapes can have properties defined in a master or individually assigned to a shape. Furthermore, a shape has a shape sheet which includes further information about the shape (e.g. position, size, etc.). The abstract meta-model is connected with the created EMF meta-model by inheritance relations. Thus, the abstract meta-model defines the complete space for Visio models and the created meta-model restrict and refines this space, respectively.

6.4.3 M1-Transformation

The M1-transformation converts models from Visio into EMF. The transformation rules depend on the previously defined M2-transformation. This transformation comprises the following rules. These rules are formally described as pseudo code in Appendix B.3, Listing B.6.

Document and Page  Starting point of the M1-transformation is a Visio document. This rule creates for each Visio document an instance of the EClass Document. Furthermore, this rule iterates over all pages and creates EObjects of the EClass Page. Both EClasses are part of the abstract Visio meta-model. The name of the Visio document and page is copied to the created EObjects.

Object Shape  This rule depends on the mapping Object Master $\mapsto$ EClass from the meta-model transformation. This rule creates an EObject for each object shape. The EClass of the created EObject corresponds the master of the current object shape. The created EObject is assigned to the previously created page EObject.

Connection Shape  This rule depends on a type mapping and corresponds to Connection Master $\mapsto$ EClass. The rule creates an EObject for each connection shape. The master of this connection shape and the EClass of the created EObject is a pair in the type mapping. The rule queries the source and target object shape and looks for the corresponding EObjects. These EObjects are assigned to the created connection EObject. Furthermore, the created connection EObject is assigned to the page EObject.

Property  Visio allows the definition of properties for masters on the language level and the definition of properties for single elements on the model level. For the first case, this rule depends on the rule Property $\mapsto$ EAttribute. The rule copies values of Visio properties to EAttributes of all model elements which conform to
6.4 Bridge 3: Microsoft Visio und Eclipse Modeling Framework

a certain master element. In the second case, this rule instantiates the EClass
Attribute and copies the name and the value of the Visio property to the created
attribute EObject. Furthermore, a shape in Visio has a default property (with the
name label). The shape EObject has also an EAttribute with the name label. This
rule copies the value of this default property to this EAttribute.

**Shape Sheet** A shape sheet represents the complete Visio model as key-values.
This rule is independent of the M2-transformation. The rule reads the shape sheet
from each Visio shape and creates a shape sheet EObject in the EMF model. Each
key-value pair is copied from the Visio shape sheet to the created EObject.

**6.4.4 Implementation**

Analogous to the other bridges, this implementation consists of the transformation
itself and some additional widgets for the input of different parameters. The current
version 1.2.0 is available at Sourceforge\(^2\) and is licensed under the Eclipse Public
License.

A general problem for the implementation of the bridge is the combination of
two different programming platforms. EMF supports Java and Visio provides .Net
languages (e.g. C#). To realize interoperability between Java and C#, we use the
Win32-Java bridge Jawin\(^3\). Based on the Visio API, Jawin generates a Java inter-
face which encapsulate the Visio API. The generated Visio-Java API can operate
on Visio stencils and models. The bridge is implemented in Java and uses the EMF
library and the generated Visio-Java API.

The first part of the bridge implements the M2-transformation. The transfor-
mation reads a selected stencil and creates a corresponding meta-model in Ecore.
The transformation implements the rules described in Section 6.4.2. The created
EMF model is serialized as XMI-Ecore file.

The second part implements the M1-transformation from Visio to EMF. The
transformation iterates over a selected Visio document, applies the transforma-
tions rules from Section 6.4.3 and produces an EMF model. This EMF model is
serialized as XMI file. The transformation considers the type mapping created by
the previously executed M2-transformation. The third part implements a trans-
formation back from EMF to Visio. The rules are analogous to the presented
transformation rules but work in the opposite direction.

**6.4.5 Example**

Analogous to the other examples, we use the EPC model again for a demonstration.
Figure 6.8(a) in the left top corner shows the EPC stencil in Visio. There are twelve
different object masters (Event, Function, XOR, AND, etc.) and one connection
master (Dynamic connector). The M2-transformation creates the corresponding
Ecore meta-model in the right top corner (Figure 6.8(b)). Figure 6.8(c) in the left

\(^2\)http://sourceforge.net/projects/visioemfbridge/
\(^3\)http://jawinproject.sourceforge.net/
bottom corner shows an EPC model in Visio. The M1-transformation creates the EMF model in the right bottom corner (Figure 6.8(d)).

Figure 6.8: Transformation example of Visio-EMF bridge
6.5 Combination of MetaEdit-EMF and Visio-EMF Bridge

In this section, we describe the combination of the Visio-EMF and MetaEdit-EMF bridge. The combination allows a migration of models between Visio and MetaEdit+ via EMF without the implementation of a further bridge. The combination of both bridges uses the approach presented in Section 4.4. Figure 6.9 illustrates the adaption of this approach to MetaEdit+ and Visio.

**Import of meta-models** Our starting point is that MetaEdit+ and Visio implement the same EPC language with slightly different meta-models. The existence of both meta-models is necessary because both bridges do not support the automatic creation of Visio or MetaEdit+ meta-models. The first step is the usage of the MetaEdit-EMF bridge transforming the MetaEdit-EPC meta-model into an EMF-EPC meta-model (denoted as $mm_{me\_emf}$). Figure 6.6(a) shows the MetaEdit-EPC meta-model as source and Figure 6.6(b) shows the created EMF-EPC meta-model as target. In the second step, the Visio-EMF bridge transforms the Visio-EPC stencil into an EMF-EPC meta-model (denoted as $mm_{visio\_emf}$). Figure 6.8(a) shows the original Visio-EPC stencil as source and Figure 6.8(b) shows the created EMF-EPC meta-model as target.

**Mapping between meta-models** Both created meta-models describe the same EPC language and are *language-equivalent*. However, both meta-models have a little bit heterogeneity stemming from different meta-modeling languages and diverse possibilities to express the EPC language. To overcome these differences between

![Figure 6.9: Combination of MetaEdit-EMF and Visio-EMF bridge](image-url)
the meta-models, we define as a third step a mapping between $mm_{visio\_emf}$ and $mm_{me\_emf}$. The mapping is expressed by the mapping language which was presented in Section 4.4.2. We implemented a mapping tool supporting this mapping language. The name of this tool is AnyMap. AnyMap uses different bridges and includes generators for the creation of transformation code. Figure 6.10 shows the user interface. The example illustrates a mapping between the two imported meta-models. The left element tree represents the MetaEdit-EPC meta-model $mm_{me\_emf}$ and the right tree represents the Visio-EPC meta-model $mm_{visio\_emf}$. The mapping in the middle consists of operators (rectangle) and links (line). For instance, the MetaEdit+ Event Driven Process Chain graph type is mapped to a Visio page because Visio does not support a graph type concept. The name property of this graph type is mapped to the generic element text of a Visio page. This property mapping is a child mapping of the graph type mapping. Both mappings are one-to-one mappings. Next, the object types (Event, Function, XOR, etc.) are mapped into the corresponding object masters. The relationship type Arc from MetaEdit+ is mapped to the Visio connection master Dynamic connector. This mapping is a many-to-one mapping because the mapping needs to include arc role types. The referenced object of the From role type is mapped to source and the object of the To role type is mapped to target of Dynamic connector.
Model migration The defined mapping is only a description and not executable. Hence, a generator reads the mapping and creates executable transformation code. Generally, the transformation code can be divided into two parts: a meta-model independent (MMI) and a meta-model dependent transformation (MMD). An MMI-transformation implements rules that are independent of a certain modeling language. These rules depend only on the participating meta-modeling tools. Once implemented, these rules can be re-used for all transformations between languages specified in the same tool. An example of an MMI-transformation rule is a transformation of symbol coordinates between MetaEdit+ symbols and Visio shapes. The second part (denoted as MMD-transformation), concerns the mapping of language-specific concepts. These transformation rules depend on particular modeling languages. Therefore, the generation of this transformation depends on the previously defined mapping. In this example, the generator reads the mapping and produces a transformation in ETL. Listing 6.1 presents a snippet of the MMD-transformation code. The first rule in Listing 6.1 corresponds to the first mapping rule in Figure 6.10. This rule creates a Visio page for each MetaEdit+ EPC graph. The name of the Visio page is set to the MetaEdit+ EPC graph’s name. The next rules describes a transformation of model elements that are instances of the Event, Function, XOR, and Arc concepts. The complete transformation code is in Appendix C.

Listing 6.1: Snippet of the generated migration transformation in ETL

```
rule graph2page
  transform event_driven_process_chain_3395083925 : INMM!Event_Driven_Process_Chain_3395083925
to evisiopage : OUTMM!EVisioPage extends Graph2Page {
evisiopage.text := event_driven_process_chain_3395083925.Name;
}

rule event2Event
  transform event_3395083771 : INMM!Event_3395083771
to event : OUTMM!Event {
event.text := event_3395083771.Name;
}

rule function2Function
  transform function_3395083784 : INMM!Function_3395083784
to function : OUTMM!Function {
  function.text := function_3395083784.Name;
}

rule xor2xor transform xor_3395083742 : INMM!XOR_3395083742
to _xor : OUTMM!XOR {}

rule arc2dynamicConnector
  transform arc_3395083800 : INMM!Arc_3395083800
to dynamic_connector : OUTMM!Dynamic_Connector {
dynamic_connector.target := arc_3395083800.me_role.equivalent();
dynamic_connector.source := arc_3395083800.me_role.equivalent();
}
```

Transformation execution  After the mapping definition and the generation of a corresponding model transformation, the migration can start. In step four, the MetaEdit-EMF bridge reads MetaEdit+ models and converts these models into equivalent EMF models ($m_{me\_emf}$). In step five, the generated model transformation produces an EMF model ($m_{visio\_emf}$) which conforms to the Visio-EMF meta-model ($mm_{visio\_emf}$). In step six, the Visio-EMF bridge converts the EMF model ($m_{visio\_emf}$) into a Visio model (document). Figure 6.11 shows input and output model of the migration transformation. Both models are equal and are expressed in the EPC language. The internal meta-models are different because of the tool-specific implementation of the EPC language.

![Figure 6.11: Transformation example for EPC models](image)
6.6 Summary

In this chapter, we applied the M3B transformation approach. We presented three bridges and a combination of two bridges. Each bridge description included the general architecture, the M2- and M1-transformation, the implementation and a short demonstration of the bridge. The demonstration showed the applicability of each bridge. Please note, that the demonstration example presents only one use case with the transformation of EPC models. The bridges also allow the migration of models expressed with other languages.

Each bridge used EMF as a target meta-modeling environment. This common meta-model environment enabled the combination of bridges. We applied a combination approach to connect two bridges. The combination allowed the migration between MetaEdit+ and Visio via EMF. The combination is supported by a tool which allows the mapping and transformation of EMF meta-models. This tool is extensible and facilitates the integration of further bridges. The combination was tested with the migration of EPC models.

With the use cases, we have come full circle since the beginning of this work, because the bridges itself and the combination of these bridges fits to the initial requirement definitions for model exchange which was described in Section 4.1.1. To conclude, this application of the bridge approach and the combination of bridges show the general applicability of this approach.
7 Evaluation of M3-Level-Based Bridges

The purpose of this chapter is to evaluate the M3B approach. The evaluation uses metrics for measuring different transformation aspects. Based on the measured results, we analyze advantages and disadvantages of the M3B approach.

7.1 Design of the Evaluation

Objective of this evaluation is the analysis of the M3B approach to understand the strengths and weaknesses of this approach. Firstly, we define a set of evaluation criteria. A good starting point for suitable evaluation criteria is the field of software quality because bridges can be regarded as software. There are various works about the quality of software. For instance, two well-known quality models in software engineering are presented by McCall et al. [1977] and Boehm et al. [1976; 1978]. Both quality models define a set of criteria and metrics for quality. Criteria are, among others, correctness, usability, maintainability, reusability or portability. Some of these criteria can be applied to the evaluation of the M3B approach.

Furthermore, there are several works focusing on the quality of model transformations. For instance, Mohagheghi and Dehlen [2007] describe a quality framework for Model-Driven Engineering. The authors discuss the quality of model transformations and describe different criteria such as performance, reusability, simplicity, and compactness. The contribution from van Amstel et al. [2008] also defines a set of quality attributes for transformations. These attributes are: understandability, modifiability, reusability, modularity, completeness, consistency, and conciseness. Furthermore, Mens and van Gorp [2006] present important requirements for model transformations such as complexity, preservation, reuse of transformations, and usability, and usefulness.

Based on the extracted criteria, we identified a set of criteria fitting our evaluation needs. These evaluation criteria are: completeness, complexity, effort, reusability, and applicability. For the analysis of these evaluation criteria, we use a conceptual argumentation. This method derives arguments for advantages and disadvantages of the transformation approach on a conceptual level. Additionally, we use metrics to measure certain transformation aspects of a concrete bridge implementation. The measured values form a basis for the conceptual argumentation and support the discussion of the evaluation criteria.

The usage of metrics requires the finding of suitable metrics. Generally, a software metric is a function which maps a defined piece of software into a numerical value. The input, output and calculation of this function depend on the spe-
specific metric. The numerical value is an indicator for a corresponding quality attribute [IEEE 1998]. Analogous to the previously selected evaluation criteria, there are metrics for program code and particular for transformation code. van Amstel et al. [2008] propose a set of transformation metrics and suggest a correlation between quality attributes and metrics. Unfortunately, the utility of these metrics is restricted because many metrics are suitable for the evaluation of transformation language concepts. But the M3B approach itself is independent of a certain transformation language. Thus, we focus metrics which are well-known and suitable for the analysis of program code. The following metrics are suitable for the evaluation and are measurable with reasonable effort: number of transformation rules, lines of code (LOC), Halstead metric, and the runtime duration of transformation executions.

The discussion about pros and cons of the M3B approach and the interpretation of the measured metric values cannot be performed without a suitable reference point. A reference point offers a way to benchmark the M3B approach in comparison to the approach used as reference point. Generally, a reference point addresses the state of the art. Hence, we take as reference point a simple model transformation (SMT). A model transformation enables the migration of models from one meta-model environment into another one, as well, but in comparison to an M3B, the implementation of a model transformation depends on specific meta-models.

A further aspect is the selection of suitable evaluation objects. The evaluation must consider two approaches: the M3B approach and as corresponding solution a simple model transformation (SMT). The evaluation of the M3B approach refers to the concept itself and a concrete bridge implementation. There are three prototypes available: ARIS-EMF, MetaEdit-EMF and Visio-EMF. We use the ARIS-EMF bridge as our evaluation object. We assume that the two other bridges have the same characteristics. The comparison between an M3B and a simple model transformation requires an equivalent transformation task. For this reason, we use a simple model transformation which transforms EPC models from ARIS to EMF. The ARIS-EMF bridge and the SMT use the same EPC meta-model and EPC models. This equality of input, output and transformation task ensures a comparability of the results.

7.2 Application of Metrics

The measurement of the selected metrics follows the same structure. We define the metric itself, describe the application of the M3B approach, and present the measured values. Analogous to the M3B measurement, we describe the application of the metric for a simple model transformation and present the measured values. After the measurement, we evaluate and compare the results of both approaches.
7.2 Application of Metrics

7.2.1 Number of Rules

Definition

This metric measures the number of transformation rules. A transformation rule is a logical unit of a transformation. This unit describes a defined set of mappings between model elements. Most transformation languages support the concept of rules explicitly. If this concept is unsupported, transformation rules must be artificially recognized by the analysis of the transformation code.

M3-Level-Based Bridge

A bridge comprises transformations at M2- and M1-level. The definition of a general formula for the calculation of transformation rules at M2-level is difficult. The implemented bridges show that the number of rules depends on the defined correspondences between the participating source and target meta-metamodel. Normally, a transformation should work without information loss. Hence, we assume that the number of rules is nearly equal to the maximum number of concepts in a source or a target meta-metamodel. Regarding the M1-transformation, the number of rules is equal to the number of rules in the M2-transformation. Additionally, there are further rules responsible for the transformation of generic concepts (e.g. structure elements or generic modeling elements).

We count the rules of the ARIS-EMF bridge. The bridge is implemented in ARIS script which does not support a rule concept, but the majority of rules is implemented as functions. However, the transformation rules of the ARIS-EMF bridge are described in Section 6.2.2 and 6.2.3. The corresponding ETL transformation is illustrated in Appendix B.1. Both transformations support only one direction, from ARIS to EMF. We count six rules for the M2-transformation and five rules for the M1-transformation. Additionally, there is one rule in the M1-transformation which is necessary for the transformation of the ARIS group structure. Overall, the ARIS-EMF bridge consists of twelve rules which are listed in Table 7.1.

<table>
<thead>
<tr>
<th>M2-transformation</th>
<th>M1-transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filter $\mapsto$ EPackage</td>
<td>–</td>
</tr>
<tr>
<td>Object Type $\mapsto$ EClass</td>
<td>Object Definition</td>
</tr>
<tr>
<td>Connection Type $\mapsto$ EClass</td>
<td>Connection Definition</td>
</tr>
<tr>
<td>Model Type $\mapsto$ EPackage and EClass</td>
<td>Model</td>
</tr>
<tr>
<td>Symbol $\mapsto$ EClass</td>
<td>Object Occurrence</td>
</tr>
<tr>
<td>Attribute $\mapsto$ EAttribute</td>
<td>–</td>
</tr>
<tr>
<td>–</td>
<td>Connection Occurrence</td>
</tr>
<tr>
<td>–</td>
<td>Group</td>
</tr>
</tbody>
</table>

Total number of rules: 12

Table 7.1: Number of rules for the ARIS-EMF bridge
Simple Model Transformation

A simple model transformation is defined against the participating meta-models. Thus, the number of transformation rules depends on a source and target meta-model. Analogous to the M3B approach, a simple model transformation should incur no information loss. Hence, we assume that the number of rules is equal to the maximum amount of concepts in a source or target meta-model.

For the measurement of the rules, we use a simple model transformation for EPC models from ARIS into EMF. This transformation reads the same input and produces the same output as the ARIS-EMF bridge. This constraint ensures the equality of both approaches. We count the rules of this simple model transformation and derive a general formula calculating the number of rules in dependency of a specific meta-model. This formula is presented in Equation 7.1. Generally, each type element in a meta-model needs a transformation rule. Hence, the number of rules is the sum of object types ($\#\text{otypes}$), connection types ($\#\text{ctypes}$), symbols ($\#\text{symbols}$) and model types ($\#\text{mtypes}$). Additional to these rules, we add two other rules. One rule is for the ARIS group transformation and the other rule is for the transformation of lines between symbols.

$$\#\text{rules} = \#\text{otypes} + \#\text{ctypes} + \#\text{symbols} + \#\text{mtypes} + 2$$  \hspace{1cm} (7.1)

Equation 7.1 can be used to calculate the number of transformation rules in dependency of a concrete meta-model. For this reason, we measured the size of existing filters (meta-models) in ARIS and apply the formula to calculate the number of rules. Table 7.2 shows the filters on the left side and the number of elements in the middle. The last two columns on the right side show the number of rules corresponding to the size of a meta-model. The total number of meta-model elements is a simple aggregation of different meta-model elements. This number can be related to the number of rules. Generally, we observed that the number of rules grows proportionally with the size of the meta-models.

<table>
<thead>
<tr>
<th>Name</th>
<th>$#\text{otypes}$</th>
<th>$#\text{ctypes}$</th>
<th>$#\text{mtypes}$</th>
<th>$#\text{symbols}$</th>
<th>Size</th>
<th>$#\text{rules}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPC</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>12</td>
<td>14</td>
</tr>
<tr>
<td>Automation filter</td>
<td>18</td>
<td>33</td>
<td>10</td>
<td>49</td>
<td>110</td>
<td>112</td>
</tr>
<tr>
<td>BPMN</td>
<td>13</td>
<td>21</td>
<td>4</td>
<td>179</td>
<td>218</td>
<td>220</td>
</tr>
<tr>
<td>Easy filter</td>
<td>27</td>
<td>47</td>
<td>11</td>
<td>181</td>
<td>266</td>
<td>268</td>
</tr>
<tr>
<td>Aris for SAP</td>
<td>22</td>
<td>57</td>
<td>16</td>
<td>179</td>
<td>274</td>
<td>276</td>
</tr>
<tr>
<td>Balanced scorecard</td>
<td>41</td>
<td>153</td>
<td>13</td>
<td>241</td>
<td>448</td>
<td>450</td>
</tr>
<tr>
<td>Business simulator</td>
<td>23</td>
<td>47</td>
<td>17</td>
<td>369</td>
<td>456</td>
<td>458</td>
</tr>
<tr>
<td>Demo database</td>
<td>66</td>
<td>151</td>
<td>43</td>
<td>262</td>
<td>522</td>
<td>524</td>
</tr>
<tr>
<td>SOA</td>
<td>96</td>
<td>227</td>
<td>24</td>
<td>499</td>
<td>846</td>
<td>848</td>
</tr>
<tr>
<td>Average</td>
<td>34</td>
<td>82</td>
<td>15</td>
<td>218</td>
<td>350</td>
<td>352</td>
</tr>
</tbody>
</table>

Table 7.2: Number of rules for simple model transformations in ARIS
Comparison

The rule number of an M3B is constant and depends on the meta-metamodels involved. The size of concrete meta-models is irrelevant. In contrast to this, the rule number of a simple model transformation (SMT) depends on a specific meta-model. The number of rules increases proportionally with the meta-model size. The diagram in Figure 7.1 illustrates the values of both approaches. The x-axis shows the meta-model size and the y-axis shows the corresponding number of rules. The red curve with rectangles represents the M3B approach and the blue curve with circles represents the usage of model transformations (SMT). There is no intersection point in this diagram. The smallest model transformation (in this case EPC) needs 14 rules and the M3B needs constantly twelve rules. Table 7.2 includes also the average number of rules. This average value represents the number of rules for the transformation of an average meta-model. The average number of rules for SMT is 352 and about 30 times greater as the twelve rules for an M3B.

![Figure 7.1: Number of rules for the ARIS-EMF bridge (M3B) and simple model transformations (SMT) in dependency of the meta-model size](image)

7.2.2 Lines of Code

Definition

Lines of code (LOC) is a common metric to measure the size of software [Kan 2002]. The LOC metric counts the number of lines in a given source code. For our case, we use the SLOC metric which counts only source code lines without comments and blank lines. SLOC is typically used to measure the effort required to develop a program. Applying the SLOC metric to a transformation is simple and requires no big effort.
M3-Level-Based Bridge

Analogous to the number of rules, we measure the SLOC of the ARIS-EMF bridge implementation. The source code of an M3B can be divided into three logical parts with different purposes. One part implements the transformation rules or logic. This part can be divided further into the M2- and M1-transformation. The second part is responsible for the creation of the abstract meta-model. The third part is infrastructure code which is responsible for loading, saving, querying, and storing mappings.

Table 7.3 shows the SLOC for the ARIS-EMF bridge. The M2-transformation needs 69 lines and the M1-transformation needs 176 lines. One reason for this difference is that the storage structure of ARIS models is more complex than the meta-model structure for ARIS filters. The infrastructure code at M2-level needs only 69 lines and at M1-level 140 lines. A reason for this difference is that the M1-transformation requires loading and saving of meta-models as well as models. A further reason is the querying mechanism to get the type of a model element and the meta-model element in the target meta-model. The creation of the abstract meta-model comprises 183 lines. Concluding, the ARIS-EMF bridge needs 637 lines of code for the whole implementation.

<table>
<thead>
<tr>
<th>Parts of M3B</th>
<th>SLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2-transformation rules</td>
<td>69</td>
</tr>
<tr>
<td>M1-transformation rules</td>
<td>176</td>
</tr>
<tr>
<td>Creation of the abstract meta-model</td>
<td>183</td>
</tr>
<tr>
<td>Infrastructure code at M2-level (save, load, etc.)</td>
<td>69</td>
</tr>
<tr>
<td>Infrastructure code at M1-level (save, load, type querying etc.)</td>
<td>140</td>
</tr>
<tr>
<td>Total</td>
<td>637</td>
</tr>
</tbody>
</table>

Table 7.3: Lines of code for the ARIS-EMF bridge

Simple Model Transformation

Analogous to the rule metric, we use the EPC model transformation and count the number of lines. Based on this measurement, we abstract from the concrete EPC meta-model and derive a formula which calculates the SLOC in dependency of a specific meta-model. Equation 7.2 shows the calculation of code lines for different meta-model concepts. The code for infrastructure (e.g. load, save, and query) is independent of a certain meta-model and comprises 140 lines. The rest of the model transformation implements mapping logic. The transformation of one symbol requires 19 lines. The transformation of one object type needs 12 lines plus one line for each attribute depending on this object type. The SLOC of one connection type is analogous and needs 23 lines plus one line per attribute. The last part is the transformation of a model type needing 21 lines plus five lines per symbol plus the number of model attributes. The transformation of attributes
is implemented as a simple assignment and requires therefore only one line per attribute.

The calculation of lines requires the number of attributes for each meta-model element and the number of symbols in each model type. To ease the overall calculation of code lines, we replace the number of attributes of an element and the number of symbols in a model type. We use the average value of attributes concerning to an element and the average value of symbols contained in a model type. In the end, the usage of the average value leads to same result. The following equations show the calculation of lines required for the transformation of an average concept in a meta-model.

\[
\begin{align*}
\text{infrastructure} &= 140 \\
\text{slines} &= 19 \\
\text{∅olines} &= 12 + \text{∅attributes} \_\text{in} \_\text{otype} \\
\text{∅clines} &= 23 + \text{∅attributes} \_\text{in} \_\text{ctype} \\
\text{∅mlines} &= 21 + 5 \times \text{∅symbols} \_\text{in} \_\text{mtype} + \text{∅attributes} \_\text{in} \_\text{mtype}
\end{align*}
\]

The summation of all parts is the total number of SLOC for a simple model transformation. Equation 7.3 show the calculation for the total number of lines. A transformation comprises the number of symbols (\#symbols) times slines plus the number of object types (\#otypes) times ∅olines plus the number of connection types (\#ctypes) times ∅clines plus the number of model types (\#mtypes) times ∅mlines.

\[
\#\text{sloc} = \text{infrastructure} + \#\text{symbols} \times \text{slines} + \#\text{otypes} \times \text{∅olines} + \#\text{ctypes} \times \text{∅clines} + \#\text{mtypes} \times \text{∅mlines}
\]

This equation can be used to calculate the lines of code in dependency of concrete meta-models. Table 7.4 shows ARIS filters and the number of elements. The left columns show the filter name and the number of elements in a filter. The columns in the middle shows the average lines of code for meta-model elements. The right columns show the aggregated meta-model size and the SLOC for a transformation. Generally, we can observe that the SLOC grows proportionally with the size of meta-models.

Comparison

The measurement shows that the M3B approach has a constant number of code lines and is independent of a specific meta-model. In contrast to this, the code lines of a model transformation depend on a meta-model. The number of lines increases proportionally with the meta-model size. This is similar to the number of rules in Section 7.2.1. The diagram in Figure 7.2 illustrates the measured values
of both approaches. The x-axis shows the size of meta-models and the y-axis is the lines of code. The red curve with rectangles illustrates the M3B approach and the blue curve with circles illustrates the simple model transformation approach (SMT). The intersection point between both curves is right at the beginning. The M3B needs constant 637 lines and a simple model transformation with the smallest meta-model (EPC meta-model) starts at 411 code lines. The average SLOC for a simple model transformation is 9579 and around 15 times greater than that of the M3B approach.

![Figure 7.2: Lines of code for the ARIS-EMF bridge (M3B) and model transformations (SMT) in dependency of the meta-model size](image-url)
7.2 Application of Metrics

7.2.3 Halstead Metric

Definition

The Halstead metric helps to estimate the complexity of software. This metric was introduced by Halstead [1977]. This metric uses the assumption that programs are composed of operators and operands. Operators and operands must be defined before the usage of the metric and depend on the programming language used. Typically, operators are keywords (e.g. function, var, return) or comparison operators (e.g ==, !=, >) and operands are typically variables, constants, or names. Furthermore, the metric distinguishes between unique operators (\( \eta_1 \)) and operands (\( \eta_2 \)) and the total number of operators (\( N_1 \)) and operands (\( N_2 \)) used in program code. Based on these measurable values, Halstead suggests formulas which allow the calculation of the following values: volume of code, difficulty, time required to program, and the number of bugs. For the evaluation, we need only the calculation of the difficulty to write or to read/understand a program. The advantage of this metric is that counting operators and operations is relatively easy. The disadvantage is that the metric concerns only the syntactic and not the semantic complexity. The formula in Equation 7.4 shows the calculation of the difficulty \( D \):

\[
D = \frac{\eta_1}{2} \times \frac{N_2}{\eta_2} .
\] (7.4)

M3-Level-Based Bridge

We apply the Halstead metric to the source code of the ARIS-EMF bridge. In the measurement, we only consider the transformation logic and exclude the code for creating the abstract meta-model and the infrastructure code. The bridge is implemented in ARIS script. Operators are, among others, function, var, if, the dot operator and typical operands are method and variable names. Table 7.5 shows the measured Halstead values for the ARIS-EMF bridge. The table is divided into four parts: rule names, values for the M2-transformation, values for the M1-transformation, and combined values for the whole bridge. The rows represent the transformation rules at M2-level and the correlating rules at M1-level (see Section 6.2). For instance, the row Object concerns the rule Object Type \( \mapsto \rightarrow \) EClass at M2-level and the rule Object Definition at M1-level. The combination of \( D_{M2} \) and \( D_{M1} \) is the sum of both values because the implementations of both transformations are different. The M2-transformation is implemented against the meta-metamodel and the M1-transformation is implemented against the data structure for models. Due to this difference, we sum both difficulties instead of using, for instance, the maximum of both values.

Generally, the calculated difficulty in Table 7.5 shows that the M1-transformation \( (D_{M1}) \) is more complex than the M2-transformation \( (D_{M2}) \). The reason for this difference is that the data structure for models is more complex than the data structure for meta-models. Furthermore, the M1-transformation must consider the mapping which was produced by the M2-transformation, but with one exception. The attribute rule at the M2-level is more complex than at the M1-level because the
transformation of attributes at M2-level must consider many different datatypes. The transformation of lines at the M2-level is not supported by the bridge, hence, the measurement of the Halstead values is impossible. Finally, the right column shows the sum of the difficulty $D_{M2}$ and $D_{M1}$. The difficulty lies between 12 and 20.

### Simple Model Transformation

The measuring object for the Halstead metric is the same EPC transformation which was previously used for the rule and LOC metric. The source code of a simple model transformation contains redundancy which is caused by repeating source code for similar meta-model concepts. For instance, the transformation of an object type into an EClass requires a certain source code. Another object type requires the same code but with slightly different changes of source and target parameters.

We assume that the transformation of more complex meta-models is analogous to the transformation of a simple meta-model. That is, the size of meta-models has a negligible influence on the complexity of simple model transformations. In this specific case, the difficulty for a simple transformation can be regarded as the difficulty for all possible model transformations.

Table 7.6 shows the measured values for a simple model transformation. In this case, we use the EPC transformation and reduce this transformation as much as possible. The difficulty for an object type with one attribute has a difficulty of 7.88. The transformation of another object type with one attribute is analogous and has the same complexity. The difficulty of object types, connection types and symbols are nearly the same value around nine. The transformation of an attribute requires

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
<th>$\eta_1$</th>
<th>$\eta_2$</th>
<th>$N_2$</th>
<th>$D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
<td>One object type with one attribute</td>
<td>7</td>
<td>28</td>
<td>63</td>
<td>7.88</td>
</tr>
<tr>
<td>Connection</td>
<td>One connection type with one attribute</td>
<td>7</td>
<td>32</td>
<td>80</td>
<td>8.75</td>
</tr>
<tr>
<td>Model</td>
<td>One model type with one attribute and one symbol</td>
<td>9</td>
<td>35</td>
<td>71</td>
<td>9.13</td>
</tr>
<tr>
<td>Symbol</td>
<td>One symbol</td>
<td>8</td>
<td>30</td>
<td>66</td>
<td>8.80</td>
</tr>
<tr>
<td>Attribute</td>
<td>One attribute</td>
<td>1</td>
<td>8</td>
<td>8</td>
<td>0.50</td>
</tr>
<tr>
<td>Line</td>
<td>One line with one connection type</td>
<td>11</td>
<td>44</td>
<td>99</td>
<td>12.38</td>
</tr>
</tbody>
</table>

Table 7.6: Halstead values for a simple model transformation
a difficulty of 0.5 because an attribute needs only an assign statement. Lines are the most difficult part because the transformation must consider a set of coordinates, a corresponding connection, and a source and target symbol.

**Comparison**

Generally, the difficulty of the M3B approach is higher than a simple model transformation (SMT). The column chart in Figure 7.3 shows the Halstead difficulty of each M3B rule (blue columns) and each class of rules in an SMT (red columns).

In more detail, all M3B rules except the line rule have a significant higher difficulty than an SMT. The reason for this difference is that the development of M3B rules requires the operation on a more abstract level than an SMT. An M3B uses a reflective API necessitating complex operations. M3B rules must cover more transformation possibilities than an SMT which is programmed for a specific meta-model. Particularly, the transformation of attributes clearly illustrates this difference. The M3B attribute rule is 33 times higher than an attribute rule in an SMT. An M3B must handle many different cases during the attribute transforma-

![Figure 7.3: Halstead difficulty of the ARIS-EMF bridge (M3B) and a simple model transformation (SMT)](image-url)
tion. In contrast to this, an SMT must consider only one specific case needing only a short assignment. The line transformation is approximately equal. The M3B considers only lines in the M1-transformation and not in the M2-transformation. Hence, both approaches use an equal transformation logic for lines. To conclude, the complexity of a bridge is about two times higher than a simple model transformation.

7.2.4 Transformation Runtime

Definition

The runtime is equal to the duration of a transformation execution. This metric investigates a dynamic aspect of both transformation approaches.

M3-Level-Based Bridge

The runtime measurement uses the SAP reference model in ARIS as dataset. This model contains over 600 models with a sufficient number of model elements. We divide the complete SAP reference model into different sub models in order to measure the runtime in dependency of the model size.

The runtime of a transformation comprises three parts: loading, transformation and saving time. The loading of models cannot be clearly measured because ARIS uses a database. There is no opportunity to recognize the loading time of ARIS models because the transformation constantly accesses the ARIS database during the whole transformation process. The saving of models is recognizable because the bridge stores the created models into an EMF file. That is, the transformation execution includes the time for loading and the time for transformation operations. The saving operation is not included in the measurement. Furthermore, the ARIS-EMF bridge comprises the M2- and M1-transformation. Thus, the runtime includes the execution time for both transformations. For the M2-transformation, we use the EPC filter. The M2-transformation returns a constant runtime because the transformation depends on the filter size and is independent of the model size. The M2-transformation needs 15 msec for the creation of a corresponding EMF meta-model. This time is vanishingly small compared to the runtime of the M1-transformation. Hence, we do not consider the M2-level transformation in the runtime calculation. The measurement is executed on a computer with the following specification: Intel Core i7-3520M CPU@2.90GHz, 16GB RAM, 64-bit Windows 7 Professional.

Table 7.7 contains the measured values of the M3B approach. The first column shows the model size. We use models with a size between 7 and 5363 elements. Additional to these models, we measure the transformation with an empty model. The runtime with model size=0 is the time which ARIS needs for the transformation initialization. We subtract this start time in order to have the time which is only relevant for the transformation itself. The second column $M3B_m$ shows the measured runtime values for the M1-transformation in milliseconds including the initialization time. Column $M3B_a$ shows the adjusted values without the initial-
7.2 Application of Metrics

ization. We can observe that the transformation time increases with the model size.

<table>
<thead>
<tr>
<th>Size</th>
<th>$M3B_m$</th>
<th>$M3B_a$</th>
<th>$SMT_m$</th>
<th>$SMT_a$</th>
<th>$\Delta_a$</th>
<th>$\frac{M3B_a}{SMT_a}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4468</td>
<td>0</td>
<td>4390</td>
<td>0</td>
<td>0</td>
<td>1.00</td>
</tr>
<tr>
<td>7</td>
<td>4486</td>
<td>18</td>
<td>4406</td>
<td>16</td>
<td>2</td>
<td>1.13</td>
</tr>
<tr>
<td>102</td>
<td>4707</td>
<td>239</td>
<td>4596</td>
<td>206</td>
<td>33</td>
<td>1.16</td>
</tr>
<tr>
<td>234</td>
<td>4987</td>
<td>519</td>
<td>4686</td>
<td>296</td>
<td>223</td>
<td>1.75</td>
</tr>
<tr>
<td>318</td>
<td>5318</td>
<td>850</td>
<td>5007</td>
<td>617</td>
<td>233</td>
<td>1.37</td>
</tr>
<tr>
<td>524</td>
<td>6229</td>
<td>1761</td>
<td>5087</td>
<td>697</td>
<td>1064</td>
<td>2.53</td>
</tr>
<tr>
<td>637</td>
<td>6249</td>
<td>1781</td>
<td>5257</td>
<td>867</td>
<td>914</td>
<td>2.05</td>
</tr>
<tr>
<td>1649</td>
<td>9474</td>
<td>5536</td>
<td>8462</td>
<td>4072</td>
<td>1464</td>
<td>1.35</td>
</tr>
<tr>
<td>2438</td>
<td>10795</td>
<td>6327</td>
<td>7891</td>
<td>3501</td>
<td>2826</td>
<td>1.81</td>
</tr>
<tr>
<td>3230</td>
<td>13530</td>
<td>9062</td>
<td>11046</td>
<td>6566</td>
<td>2406</td>
<td>1.36</td>
</tr>
<tr>
<td>5363</td>
<td>17005</td>
<td>12537</td>
<td>10705</td>
<td>6315</td>
<td>6222</td>
<td>1.99</td>
</tr>
</tbody>
</table>

Table 7.7: Runtime in msec of the ARIS-EMF bridge (M3B) and a simple model transformation (SMT) in dependency of the model size

**Simple Model Transformation**

The evaluation of the model transformation investigates the same EPC transformation used in the previous metrics. The transformation is programmed against the EPC filter, gets executed on the same computer, and uses the same models as the M1-transformation from the ARIS-EMF bridge. Furthermore, the transformation can be divided into the same three parts: loading, transformation operations and saving. Analogous to the M3B measurement, the loading of models is part of the transformation time.

The results of the measurement are presented in the fourth column $SMT_m$ of Table 7.7. The transformation of an empty model (size=0) leads to a runtime of 4390 msec. We subtract this initialization time from all other measured values and get the runtime for the transformation without initialization time but with loading time for elements. Column $SMT_a$ shows these adjusted values. We can observe that the transformation time grows with the model size. The smallest model with seven elements needs 16 msec and the biggest model with 5363 element needs 6315 msec.

**Comparison**

Figure 7.4 illustrates the runtime of both approaches. The red curve with rectangles shows the M1-transformation as part of the M3B and the blue curve with circles shows an equivalent model transformation (SMT). The time of both transformations grows with the model size. Generally, the M1-transformation needs longer than the SMT. The sixth column $\Delta_a$ in Table 7.7 shows the absolute difference in milliseconds and the last column $\frac{M3B_a}{SMT_a}$ shows the ratio between both transformation approaches. In this case, the SMT serves as reference point with a value of
100%. The M1-transformation needs 113% for a small model (size=7), 181% for a medium model (size=2,438) and 199% for the biggest model (size=5,363). That is, the M1-transformation needs for a big model twice as long as the SMT. We assume that the runtime difference between M3B and SMT increase with the model size.

The following reasons could cause this difference. The M1-transformation uses a reflective API for querying the type of model elements. The creation of an EMF model uses the dynamic creation approach with a reflective API. Furthermore, the M1-transformation must look for corresponding types in the mapping table. We assume that the usage of the reflective API and the additional operations increase the time for the M1-transformation.

### 7.3 Evaluation Criteria

The purpose of the evaluation is the discussion of selected criteria to provide an objective argumentation about the advantages and disadvantages of the M3B approach. The evaluation criteria are completeness, complexity, effort, reusability, and applicability. The foundation of this evaluation is the M3B approach itself (see Section 4), experiences about the development and application of different M3Bs (see Section 6), and the results of the measured metrics. Similar to the metrics, we use the concept of a simple model transformation (SMT) as reference point in our argumentation because an SMT is state of the art in implementing migrations of models between meta-modeling environments.
7.3 Evaluation Criteria

7.3.1 Completeness
The completeness of the M3B approach refers to the M2- and M1-transformation. Both transformations should migrate meta-models or models without a loss. We assume that the higher the transformation loss, the lower the completeness. The transformation loss can be determined with the help of the source and target models. Both models should be expressed with the same language but in different meta-modeling environments. Furthermore, both models should be equivalent. A transformation loss can include a loss of model elements, a loss of structure or a loss of syntactical/graphical issues.

Based on our use cases, we observed that all three bridges produce suitable meta-models and models in the target environment. The created target models are almost equal with the source model. In comparison to an SMT, the application of a bridge produces the same transformation result with the same completeness of models between a source and target. Hence, the bridge approach itself is not responsible for a possible transformation loss. In fact, the reason for an information loss is the heterogeneity between meta-metamodels.

Regarding the M2-transformation, the reason for an information loss is caused by different meta-modeling concepts which cannot be mapped onto each other. There are three possible scenarios. The first scenario is a simple mapping between two meta-modeling concepts. This mapping is often a one-to-one mapping between equal concepts. This scenario leads to no loss of information. The second scenario is a complex mapping. In this case, a source concept must be emulated by a set of different target concepts. This mapping can lead to an information loss between meta-models. The last scenario is that a source concept cannot be expressed by a target concept. This leads to an information loss on the meta-model level.

The information loss of the M1-transformation depends on the M2-transformation. In case of the first scenario, the M1-transformation can migrate models without information loss. Regarding the second scenario, the M1-transformation can lead to an information loss. The third scenario leads to an information loss because there are no corresponding meta-model elements. A further possibility for an information loss at M1-level can be a missing mapping between generic model elements within the model storage structure.

Concluding, the completeness of the transformations is independent of the M3B approach itself. M3Bs produce the same result as an SMT. The reason for a lack of completeness is the heterogeneity between the participating meta-metamodels.

7.3.2 Complexity
Complexity deals with the difficulty of developing and using transformations. Generally, we can state that the development complexity of the M3B approach is higher than an SMT. An SMT is developed for a specific meta-model. In contrast to an SMT, an M3B is developed against a source and target meta-metamodel. The change in abstraction level leads to higher complexity because an M3B must consider all possible meta-models. Due to the larger state space, M3B rules must cover more possible cases than SMT rules. Furthermore, the coupling of both transfor-
mations also increases the complexity in comparison to an SMT. Instead of one model transformation, an M3B comprises two transformations which depend on each other.

The Halstead metric supports this statement about complexity. The measured values indicates that the difficulty of an M3B is higher than an SMT. Figure 7.3 illustrates that each rule in an M3B except the line transformation is more difficult than rules in an SMT. Although, this metric concerns only the syntactical aspects and not the semantics of transformations, we assume that the syntactical aspect influences the semantic aspect. Besides this, we assume that the semantic complexity of the M3B approach is higher than an SMT because the higher abstraction level. An M3B requires the usage of reflective programming and as already mentioned a coupling of two transformations.

From a user viewpoint, the complexity between the M3B approach and an SMT is almost the same. Both approaches can be used without programming skills. The usage of an SMT only requires the selection of a source model and the execution of the transformation. The M3B bridge requires the selection of a source model and meta-model as well as the execution of two transformations: one for meta-models and one for models.

7.3.3 Effort

The effort describes the amount of work which is necessary for transformation development. The M3B approach requires an effort only once for the implementation of the M2- and M1-transformation. The development effort depends on the complexity of the meta-metamodels and not on specific meta-models. An SMT requires an implementation for each meta-model. The effort depends on the complexity of the meta-model. Thus, the development effort for a bridge is constant and the effort for the SMT increases with the number of meta-models considered.

Both metrics, lines of code and number rules confirm this statement. Interestingly the intersection point between both approaches is at the beginning of the curves for both metrics, meaning the amount of code or rules for an M3B amortize after nearly one transformation between a simple meta-model with a size of circa seven elements.

Unfortunately, the investigated metrics make no statement about the time which is also a factor for the effort. Generally, there is a correlation between the number of lines/rules and the time for programming these lines/rules. Furthermore, the time for programming depends also on the complexity of writing a program. We cannot make an exact statement about this time. We assume that the development of an SMT is faster than the development of a bridge because a bridge has a higher complexity than an SMT. But there must be also an intersection point because the time for programming an SMT depends on the meta-model size and increases with each new development. To conclude, the initial development effort of a bridge is higher than of an SMT, but the development of many SMTs requires more effort over time than the development of one bridge.
Besides the development effort, we investigate the execution effort, particular the runtime. The runtime of an M3B comprises the runtime of the M2- and M1-transformation. The runtime of the M2-transformation depends on the meta-model size and is independent of the model size. The runtime of the M2-transformation is very short in comparison to the M1-transformation. Therefore, we only consider the M1-transformation. Figure 7.4 illustrates the runtime of both approaches in dependency of the model size. We observed that the runtime of the M1-transformation increases faster than the runtime of an SMT. The runtime difference increases with the model size. The longer runtime of a M1-transformation is a result of the execution of more complex operation than in an SMT.

7.3.4 Reusability

Reuse is an important aspect in software development. For instance, reuse can reduce the implementation effort. Instead of re-implementing software again and again, reusable software can be easily applied or adapted to other scenarios. The M3B approach can be applied to different meta-models. A bridge reuses general transformation logic necessary for the migration. This reuse is possible because the bridge is implemented against the participating meta-metamodels. An SMT cannot reuse transformation logic like the bridge approach. Typically, an SMT has a specific transformation logic which is implemented against specific meta-models. Nevertheless, some transformation languages support reuse concepts in their language definition. But these language concepts are not comparable to the underlying transformation approach.

Furthermore, reuse has a significant impact on software quality because frequent use of software helps to find possible errors. In this case, the argumentation is similar to the discussion about the effort. A bridge can be re-used for each meta-model. The reuse helps to find failures which can lead to an improvement of quality for later transformations with other meta-models. In contrast to this, an SMT is implemented for a narrow application space. The application of an SMT is focused on a specific modeling language. The transformation of many different languages requires a new SMT. The development of a new transformation increases the probability for errors which leads to worse quality.

To conclude, in context of the defined migration task, the reusability of an M3B is higher than an SMT. This higher reuse can improve the transformation quality and can lead to less implementation effort.

7.3.5 Applicability

The developed M3B approach is suitable for a specific transformation task. An M3B allows only the migration of models between different meta-modeling environments. Thus, the application field of M3Bs is limited in comparison to an SMT. An SMT provides a greater application spectrum (e.g. migration, synthesis, merging, analysis, refinement). Nevertheless, the specialization of the M3B approach to one specific task results in some advantages (e.g. reduction of implementation effort, higher reusability, higher quality).
The M3B approach is not a theoretical construct. The approach was applied in the implementation of three bridges between different tools. These tools are used in practice which emphasize the applicability of the M3B approach. The application of this approach requires meta-modeling tools with a (classical) three-layer meta-modeling structure. We do not apply the approach to other meta-modeling architectures. Furthermore, the bridge considers only transformation rules between meta-modeling concepts contained in the investigated meta-modeling languages. We cannot say anything about other meta-modeling concepts.

Generally, we observed a high applicability of this approach in compliance with defined conditions. These conditions are mainly defined by the transformation task and the structure of the meta-modeling environments.

7.4 Summary

In this chapter, we evaluated the M3B approach. The evaluation was divided into two parts. The first part used metrics for the measurement of certain properties. The second part used the results from the first part and discussed some evaluation aspects such as completeness, complexity, effort, reusability, and applicability. The evaluation used the concept of an SMT as a reference point. This reference point allows a comparison of both approaches, thus easing the interpretation of the results.

The application of metrics was reasonable. The selection of these metrics and the measurement was a challenging task. There are a lot of more metrics which could be interesting for this evaluation, but the application of further metrics would increase effort without yielding results of reasonable proportions. However, the selected metrics gave a good foundation for the evaluation. Beyond this work, these metrics can be re-used in similar works relating to other transformation approaches.

The evaluation has shown the high applicability of the bridge concept in compliance with the defined migration problem. The completeness is sufficient to produce suitable transformation results. The complexity of a bridge is higher than that of an SMT. Regarding the model hierarchy, a bridge uses a higher abstraction level than an SMT and a bridge consists of two coupled transformations. The initial effort for the development of a bridge is higher than that of an SMT, but the bridge covers more transformation cases than an SMT. The reusability of a bridge is higher than that of an SMT. This has a positive effect on the quality of bridge transformations.
8 Conclusion and Future Work

8.1 Summary and Conclusion

8.1.1 Interoperability between Meta-Modeling Environments

This work addressed the problem of missing model interoperability between modeling tools. Interoperability is necessary for building tool chains in complex software development processes. Furthermore, interoperability is required for replacing a tool by another one, better fitting customer’s needs. There are many aspects in the area of interoperability. This thesis focused on the migration of models between different meta-modeling tools implementing a three-level model hierarchy.

Despite many existing approaches, the exchange of models was not solved sufficiently. This finding resulted from a study conducted as part of this thesis. The study investigated the exchange of models and meta-models between different tools. The study surveyed over 60 modeling tools and investigated the exchange mechanism of 20 meta-modeling tools in more detail. Only less than one percent of all possible connections between the participating tools allow an exchange of models as well as meta-models. This marginal number of exchange possibilities was unsatisfying. This thesis tried to close the gap of missing interoperability with the development of a suitable exchange approach.

Generally, the exchange of models requires to overcome heterogeneity between different meta-modeling environments. There are basically two approaches to tackle the heterogeneity problem. One approach is the usage of a common and standardized meta-model hierarchy serving as a unifying element between different meta-modeling environments. The second mechanism is the usage of transformations to map different meta-model hierarchies onto each other. Both approaches have their advantages and disadvantages. In this thesis, we followed the second approach and developed a solution to achieve interoperability.

The first contribution of this work is the development of a transformation approach. This approach is denoted as M3-level-based bridge and enables the exchange of models and meta-model between different model hierarchies. The M3B approach was successfully applied to the implementation of three bridges. These bridges are between ARIS and EMF, MetaEdit+ and EMF, and Visio and EMF. These implementations showed that the approach is suitable for the usage in a practical context. The Visio-EMF bridge achieved the biggest impact of all three bridges. One reason could be the prevalence of Visio in comparison to MetaEdit+ or ARIS. Another reason could be the missing capability for processing of Visio models. For instance, Visio offers no special language for developing code generators or transformations. The Visio-EMF bridge enables the processing of Visio models with EMF tools. Thus, this feature satisfies a demand of many Visio users. All
implemented bridges have a prototype character. Nevertheless, all bridges showed the feasibility of the presented approach.

To conclude, the implemented bridges increase the interoperability between the participating tools. Furthermore, we assume that the approach is suitable to increase interoperability between further meta-modeling tools.

8.1.2 Transformation of Models and Meta-Models

The second contribution refers to the aspect of model transformations. In this thesis, we formalized the M3B approach and classified the approach with the help of existing transformation characteristics. An M3B comprises two coupled transformations: an M2-transformation for the migration of meta-models and an M1-transformation for the migration of models. The M1-transformation consists of a fix part responsible for a generic transformation between models. This fix part is implemented against the data structure of models. The second part of the M1-transformation depends on a mapping between meta-models previously created by the M2-transformation. The M1-transformation reads and interprets the mapping to control the creation of target models.

The M1-transformation is a higher-order transformation. Higher-order transformations are a current topic in research. The M3B approach is a good application for higher-order transformations and provides valuable information for this kind of transformation. Additional to the formalized description, the thesis discussed the development of bridges. We introduced a general transformation system, an abstract development process and some technical implementation aspects. Beyond that, we presented the combination of bridges and suggested a mapping language for the specification of mappings between language-equivalent meta-models.

Further important objectives of this thesis were the application and evaluation of the M3B approach. The uses cases demonstrated the general application of the transformation approach. The evaluation showed the advantages and disadvantages in more detail. During the evaluation, we measured selected metrics and compared the bridge approach to an equivalent model transformation. The advantage of the M3B approach is that the transformation takes place on a higher abstraction level. This abstraction enables the transformation of models in dependency of their meta-models without changing the transformation implementation. In contrast to the M3B approach, a simple model transformation requires a specific implementation for each meta-model. Hence, the effort of a simple model transformation increases with each application but the effort for the development of an M3B is constant. Furthermore, the evaluation confirmed that the development of the bridge is more complex than a simple model transformation.

8.1.3 Understanding of Meta-Modeling Languages

The third contribution of this thesis concerns the understanding of meta-modeling languages. The mapping between meta-modeling languages is a central issue of an M3B. We investigated different languages and extracted their concepts. Based on this description, we developed a common framework consisting of meta-modeling
8.2 Directions for Future Research

8.2.1 Interoperability between Meta-Modeling Environments

An M3B allows the integration of meta-modeling tools with a classical three-level meta-modeling hierarchy. Additional to the three-layer hierarchy, there are further meta-modeling mechanisms such as lightweight meta-modeling [Frankel 2003] or multi-level modeling [Atkinson and Kühne 2001; Frank 2014]. A future work may research the extension of the bridging approach to cover further meta-modeling architectures. Additional to the pure meta-modeling domain, there are other technologies which also support a three-level structure (explicitly or implicitly). For instance, Excel supports a three-level hierarchy consisting of the Excel table specification (M3-level), specific tables with column names (M2-level), and values in these tables (M1-level). A connection between meta-modeling environments and other technologies would allow an enrichment of models with data from systems running in the real world (e.g. information systems). This kind of combination can bring benefits, for instance, in the field of simulation or reporting.
A further open issue concerns the transformation process. The M3B approach itself concentrates only on the mapping aspect between different meta-model environments and implements the underlying transformation process as simple as possible. But in practice, bridges must manage more complex scenarios. One issue concerns the synchronization of models. Currently all implemented bridges can export or import models in one piece. There is no synchronization mechanism allowing the exchange of certain model elements. Another aspect concerns the model quality during import or export. Currently, bridges do not support the validation of models against defined quality criteria. This missing feature can lead to errors during the transformation process. The development of a synchronization and validation mechanism could be further research tasks.

The mapping of meta-modeling languages was focused only on the abstract syntax (meta-model). This work did not investigate the transformation of the concrete syntax. The implemented bridges use a rudimentary solution for the concrete syntax and transform only the position and size of model elements. This is sufficient for re-creating models in the target environment. A future work may be the investigation of different languages for the definition of concrete syntax and the suggestion for mappings between these languages. This mapping could be used to extend the bridging approach.

Besides the M3B approach, the thesis introduced an approach allowing the combination of bridges. The advantage of this combination approach is the connection of meta-modeling tools through the reuse of existing bridges. This reuse minimizes the effort required for the development of new connections between tools. Future research may address this combination approach. The proposed approach was applied successfully to the combination of the MetaEdit-EMF and Visio-EMF bridge. However, the combination of further bridges would be interesting in order to improve this approach. A possible task may concern the expressiveness and usability of the proposed mapping language. Further research could address the whole transformation process including the registration and management of different bridges, the management of generators, and the development of a user interface as part of a common bridging platform.

Generally, we conclude that the exchange of models and integration of modeling tools is a continuous task. Although there is a large number of works focusing on this area, there will be still a need for further research. This statement can be supported by an expert group which discussed issues related to domain-specific languages on a Dagstuhl workshop. One point on their research agenda is the integration of domain-specific languages and “opening of tools to provide access for others and hooking tools into other tools without standardization” [Erdweg et al. 2015].

8.2.2 Automation and Reuse of Model Transformations

The combination of bridges requires a mapping between language-equivalent meta-models (see Section 4.4). The mapping is necessary to overcome heterogeneity between meta-models. Based on this mapping, a generator produces an executable
transformation. The creation of mappings is a repeating task for each combination of meta-models. At the moment, each mapping must be manually created during the exchange process. Hence, the creation of mappings can be time-consuming and error-prone. A future work could be the automatic calculation of mappings. The calculated suggestions may assist in mapping development and increase development speed and quality.

Calculation of mappings is a challenging task with different strategies to solve this problem. We suggest an approach which reuses knowledge from existing mappings [Kern et al. 2015a; Dimitrieski et al. 2015]. The proposed reuse approach comprises a repository and different reuse algorithms. The repository stores previously created mappings. The reuse algorithm compares elements from the current mapping scenario with existing mapping rules in the repository. The result of an reuse algorithm is a set of rules with probabilities indicating the suitability for the current mapping scenario. The algorithm must adapt the suggested mapping rules to the current mapping problem and compose a valid mapping for a complete meta-model. The assumption is that the reuse approach produces results with a higher quality than a mapping approach which calculates mappings from scratch without background knowledge. A prove of this statement is future work. Another issue for future work is the composition of mapping rules to a valid mapping.

8.2.3 Research on Meta-Modeling

The analysis of meta-modeling languages considered seven languages. The extracted framework can be used for the analysis of further meta-modeling languages. An extension of this investigation with further meta-modeling languages may improve the quality of the framework. Possible meta-modeling languages are, among others, MEMO [Frank 2011], ADOxx [Fill et al. 2013], MetaDONE [Englebert and Magusiak 2013], Melanee [Atkinson and Gerbig 2012], MetaDepth [de Lara and Guerra 2010] or Diagram Predicate Framework (DPF) [Lamo et al. 2012]. A further issue, besides the extension of the framework, is an improvement of the formalization. Currently, the framework lacks in the formalization of concepts. Fill et al. [2012] provide a formalism for describing ADOxx meta-models and models. This formalism could be a suitable starting point for a possible formalization. Although the suggested formalization approach was used only for the ADOxx, we could imagine to apply this formalization to the concept framework of this thesis.

Another interesting issue for future work is an investigation into the expressiveness of different meta-modeling languages. The thesis has not explicitly investigated the expressiveness. We have seen that the transformation between different meta-modeling languages can lead to an information loss. This fact shows that there are language concepts with different expressiveness. For instance, there are different variants of relationships which may influence the expressiveness for meta-modeling languages. We could imagine something similar to a Chomsky hierarchy for meta-modeling languages. The investigation of expressiveness can improve the theoretical understanding of meta-modeling languages which in turn can influence
the development of languages and the application of meta-modeling tools in the future.

Besides the theoretical expressiveness of meta-modeling languages, an investigation about the practical usability of meta-modeling concepts may be an interesting task. Currently there are good and powerful meta-modeling languages established in the real world. But a self-reflective and empirical investigation about existing meta-models and meta-modeling languages could lead to findings which may improve existing languages. For instance, Visio offers a simple meta-modeling language with only three main concepts. On the one hand, the definition of Visio modeling languages is intuitive and sufficient. But on the other hand, the usage of Visio offers a large free space to create models. This free space can lead to a poor quality of models and hinder the processing of Visio models because corresponding language definitions are may be too weak. In contrast to Visio, MetaEdit+ offers more concepts, thus enabling a more precise definition of modeling languages. However, Visio and MetaEdit+ allow the creation of almost identical models but the higher expressiveness of MetaEdit+ eases the creation and processing of models. The investigation of theoretical expressiveness and practical usability is an essential issue, particularly for the meta-modeling approach.
## A Meta-Modeling Tools and Exchange Formats

Table A.1 shows a list of 69 modeling tools. Based on this list, we select 20 meta-modeling tools for the study about interoperability in Section 3. The last column indicates the meta-modeling capabilities of each tool.

<table>
<thead>
<tr>
<th>Name</th>
<th>Vendor</th>
<th>Version</th>
<th>Meta-modeling capabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agilian</td>
<td>Visual Paradigm</td>
<td>4</td>
<td>•/•</td>
</tr>
<tr>
<td>Altova UModel</td>
<td>Altova</td>
<td>2012</td>
<td>•/–</td>
</tr>
<tr>
<td>ArgoUML</td>
<td></td>
<td>0.34</td>
<td>•/–</td>
</tr>
<tr>
<td>Archi</td>
<td>University of Bolton</td>
<td>2.3</td>
<td>–/–</td>
</tr>
<tr>
<td>ARIS Business Architect</td>
<td>Software AG</td>
<td>7.1</td>
<td>•/•</td>
</tr>
<tr>
<td>ARIS Express</td>
<td>Software AG</td>
<td>2.3</td>
<td>–/–</td>
</tr>
<tr>
<td>Artisan Studio</td>
<td>Atego</td>
<td>7.4</td>
<td>•/–</td>
</tr>
<tr>
<td>Astah</td>
<td>Atego</td>
<td>6.6.3</td>
<td>•/–</td>
</tr>
<tr>
<td>AToM3 McGill University</td>
<td>McGill University</td>
<td>2008</td>
<td>–/•</td>
</tr>
<tr>
<td>bflow* Toolbox</td>
<td></td>
<td>1.2.5a</td>
<td>–/–</td>
</tr>
<tr>
<td>Bizagi Process Modeler</td>
<td>Bizagi</td>
<td>2.3</td>
<td>–/–</td>
</tr>
<tr>
<td>BOUML</td>
<td>Bruno Pagès</td>
<td></td>
<td>–/–</td>
</tr>
<tr>
<td>Cadifra UML Editor</td>
<td>A. &amp; F. Buehlmann</td>
<td>1.3.3</td>
<td>–/–</td>
</tr>
<tr>
<td>CaseComplete</td>
<td>Serlio Software</td>
<td>7.0 (2012)</td>
<td>–/–</td>
</tr>
<tr>
<td>ConceptDraw</td>
<td>CS Odessa</td>
<td>9</td>
<td>–/•</td>
</tr>
<tr>
<td>Cubetto Toolset</td>
<td>Semture</td>
<td>1.7.1</td>
<td>–/•</td>
</tr>
<tr>
<td>Database Design Tool</td>
<td></td>
<td>1.5</td>
<td>–/–</td>
</tr>
<tr>
<td>DB Wrench</td>
<td>Nizana Systems</td>
<td>2.3.0</td>
<td>–/–</td>
</tr>
<tr>
<td>dbConstructor</td>
<td>DBDeveloper Solutions</td>
<td></td>
<td>–/–</td>
</tr>
<tr>
<td>DbSchema</td>
<td>Wise Coders Solutions</td>
<td></td>
<td>–/–</td>
</tr>
<tr>
<td>Dia</td>
<td></td>
<td>0.97.2</td>
<td>–/•</td>
</tr>
<tr>
<td>Edraw Max</td>
<td>EdrawSoft</td>
<td>6.3</td>
<td>–/•</td>
</tr>
</tbody>
</table>

Table A.1: Modeling tools – continued on next page
<table>
<thead>
<tr>
<th>Name</th>
<th>Vendor</th>
<th>Version</th>
<th>Meta-modeling (light/heavy)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enterprise Architect</td>
<td>Sparx Systems</td>
<td>9.3</td>
<td>●/●</td>
</tr>
<tr>
<td>ER Creator</td>
<td>modelCreator Software</td>
<td>3.0</td>
<td>−/−</td>
</tr>
<tr>
<td>ER/Studio Software Architect</td>
<td>Embarcadero Technologies</td>
<td>1.1.0</td>
<td>●/−</td>
</tr>
<tr>
<td>ER/Studio Business Architect</td>
<td>Embarcadero Technologies</td>
<td>1.7.0</td>
<td>−/−</td>
</tr>
<tr>
<td>Generic Modeling Environment</td>
<td>Vanderbilt University</td>
<td>10.8</td>
<td>−/●</td>
</tr>
<tr>
<td>Gliffy</td>
<td>Gliffy</td>
<td></td>
<td>−/−</td>
</tr>
<tr>
<td>Grapholite</td>
<td>Perpetuum Software</td>
<td>1.6.0.7</td>
<td>−/−</td>
</tr>
<tr>
<td>iGrafix Process</td>
<td>iGrafix</td>
<td>2011</td>
<td>−/●</td>
</tr>
<tr>
<td>Intalio BPMS Designer</td>
<td>Intalio</td>
<td>6.1.12</td>
<td>−/−</td>
</tr>
<tr>
<td>Lucidchart</td>
<td>Lucid Software</td>
<td></td>
<td>−/●</td>
</tr>
<tr>
<td>MagicDraw</td>
<td>NoMagic</td>
<td>17.0.2</td>
<td>●/−</td>
</tr>
<tr>
<td>Maram Meta-Tools</td>
<td>University of Auckland</td>
<td></td>
<td>−/●</td>
</tr>
<tr>
<td>MetaEdit+</td>
<td>MetaCase</td>
<td>5.0</td>
<td>−/●</td>
</tr>
<tr>
<td>Microsoft Visio</td>
<td>Microsoft</td>
<td>2010 (14)</td>
<td>−/●</td>
</tr>
<tr>
<td>Modelio</td>
<td>Modeliosoft</td>
<td>2.1.1</td>
<td>●/−</td>
</tr>
<tr>
<td>NClass</td>
<td>Balazs Tihanyi</td>
<td>2.04</td>
<td>−/−</td>
</tr>
<tr>
<td>Objecteering</td>
<td>Objecteering Software</td>
<td>6.1</td>
<td>●/−</td>
</tr>
<tr>
<td>objectiF</td>
<td>microTOOL</td>
<td>7.1</td>
<td>●/−</td>
</tr>
<tr>
<td>Open ModelSphere</td>
<td>Grandite</td>
<td>3.2</td>
<td>●/−</td>
</tr>
<tr>
<td>ORM Designer</td>
<td>Inventic</td>
<td></td>
<td>−/−</td>
</tr>
<tr>
<td>Poseidon for UML</td>
<td>Gentleware</td>
<td>8</td>
<td>−/−</td>
</tr>
<tr>
<td>Papyrus</td>
<td>Eclipse</td>
<td>1.12</td>
<td>●/−</td>
</tr>
<tr>
<td>PowerDesigner</td>
<td>Sybase</td>
<td>16.1</td>
<td>●/●</td>
</tr>
<tr>
<td>Process Modeler</td>
<td>itp commerce</td>
<td>5</td>
<td>−/−</td>
</tr>
<tr>
<td>RISE</td>
<td>RISE to Bloome Software</td>
<td>4.5</td>
<td>−/−</td>
</tr>
<tr>
<td>Select Architect</td>
<td>Select Business</td>
<td></td>
<td>●/−</td>
</tr>
<tr>
<td>SemiTalk</td>
<td>Semtation</td>
<td>4</td>
<td>−/−</td>
</tr>
<tr>
<td>Signavio Process</td>
<td>Signavio</td>
<td>6.0</td>
<td>−/−</td>
</tr>
<tr>
<td>SmartDraw</td>
<td>SmartDraw Software</td>
<td></td>
<td>−/−</td>
</tr>
<tr>
<td>Topcased</td>
<td></td>
<td>5.2</td>
<td>●/−</td>
</tr>
<tr>
<td>UML Lab</td>
<td>Yatta Solutions</td>
<td>1.4.3</td>
<td>●/−</td>
</tr>
</tbody>
</table>

Table A.1: Modeling tools – continued on next page
<table>
<thead>
<tr>
<th>Name</th>
<th>Vendor</th>
<th>Version</th>
<th>Meta-modeling (light/heavy)</th>
</tr>
</thead>
<tbody>
<tr>
<td>UMLet</td>
<td></td>
<td>11.5.1</td>
<td>−/−</td>
</tr>
<tr>
<td>ViFlow</td>
<td>ViCon</td>
<td>0.21.1</td>
<td>−/•</td>
</tr>
<tr>
<td>Violet UML Editor</td>
<td></td>
<td></td>
<td>−/−</td>
</tr>
<tr>
<td>Visual Paradigm for UML</td>
<td>Visual Paradigm</td>
<td>9</td>
<td>•/•</td>
</tr>
<tr>
<td>Visualization and Modeling SDK</td>
<td>Microsoft</td>
<td>VS2012</td>
<td>−/•</td>
</tr>
<tr>
<td>WinA&amp;D</td>
<td>Excel Software</td>
<td></td>
<td>−/−</td>
</tr>
<tr>
<td>Xcase</td>
<td>Resolution Software</td>
<td>9.1</td>
<td>−/−</td>
</tr>
<tr>
<td>yED</td>
<td>yWorks</td>
<td>3.9.2</td>
<td>−/•</td>
</tr>
</tbody>
</table>

Table A.1: Modeling tools – continued from previous page

Table A.2 shows the import and export formats of tools included in the interoperability study.

**Agilian**
- **Import**: Rational Rose (mdl) files, Rational DNX files, BizAgi project file, specific XML, XMI (1.2, 2.1), Eclipse UML2 (XMI 2.1), Visual Paradigm project file, MS Excel file with specific schema, Visio ERD, Visio drawing/stencils into Agilian Stencil, NetBeans 6.x UML diagrams, Telelogic System Architect, Telelogic Rhapsody, PowerDesigner project file
- **Export**: BPMN2.0-XML, specific XML, XMI (1.2, 2.1), Eclipse UML2 (XMI 2.1), Visual Paradigm project file, MS Excel file with specific schema, VPP (ZIP project archive)

**ARIS Business Architect**
- **Import**: XML with specific schema, UML (XMI1.1)
- **Export**: ADF (ARIS filter), XMI, XML, Visio (VDX), BPEL, ADB (ARIS database)

**Business Process Visual Architect**
- **Import**: BizAgi project file, XML, BPMN2.0-XML, XPDL2.1, Telelogic System Architect, Excel, Visio
- **Export**: BPMN 2.0 XML, XML, BPMN2.0-XML, XPDL2.1, Excel

**ConceptDraw**
- **Import**: Visio (VDX), MS PowerPoint
- **Export**: CDX file (XML), Visio (VDX), MS PowerPoint

Table A.2: Import and export formats of meta-modeling tools – continued on next page
<table>
<thead>
<tr>
<th>Cubetto Toolset</th>
<th>Import –</th>
<th>Export ETZ format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dia</td>
<td>Import Visio models, specific XML file</td>
<td>Export Visio models, specific XML file</td>
</tr>
<tr>
<td>Edraw Max</td>
<td>Import Visio</td>
<td>Export</td>
</tr>
<tr>
<td>Enterprise Architect</td>
<td>Import Database Schema, specific Visio models (Communication, Activity, Class, Object, Component, Deployment, Custom), Doors, XMI (UML 1.1, 1.3 or 2.x), ARCGIS, ODM (OWL/RDF), Rhapsody, Rational Software Architect (EMX/UML2)</td>
<td>Export XMI 1.0 (UML1.3), XMI 1.1 (UML1.3), XMI 1.2 (UML1.4), XML 2.1 (UML2.0), MOF1.4 (XMI1.2), MOF1.3 (XMI1.1), specific XML, Ecore, OWL/RDF, BPMN2-XML</td>
</tr>
<tr>
<td>iGrafix Process</td>
<td>Import Visio models and metamodels</td>
<td>Export BPEL XML, XPDL, XML</td>
</tr>
<tr>
<td>Lucidchart</td>
<td>Import Visio models and metamodels</td>
<td>Export Visio models and metamodels</td>
</tr>
<tr>
<td>MetaEdit+</td>
<td>Import GXL-adapted (models and metamodels)</td>
<td>Export GXL-adapted (models and metamodels)</td>
</tr>
<tr>
<td>Microsoft Visio</td>
<td>Import –</td>
<td>Export –</td>
</tr>
<tr>
<td>PowerDesigner</td>
<td>Import Excel, ERwin, XMI, Rational Rose (MDL), SIMUL8 file, specific Visio models</td>
<td>Export UML2, XMI2.1 XML schema files</td>
</tr>
<tr>
<td>Visual Paradigm for UML</td>
<td>Import ERWin Data Modeler project files, BizAgi project file, System Architect business process diagram, XMI (1.2, 2.1), Excel, Visio ERD, Visio diagram to Stencil, Rational Rose (MDL) files, Rational DNX files, Rational Software Architect files, PowerDesigner project file, Telelogic Modeler</td>
<td>Export BPEL, XPDL, JPDL, BPMN2.0-XML, XMI (1.2, 2.1), Excel, SCXML</td>
</tr>
</tbody>
</table>

Table A.2: Import and export formats of meta-modeling tools – continued on next page
<table>
<thead>
<tr>
<th>yEd</th>
<th>Import</th>
<th>Export</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Graph Markup Language (GRAPHML), yWorks Binary Graph Format, Graph Modeling Language (GML, XGML), Trivial Graph Format (TGF), Gedcom Data (GED)</td>
<td></td>
</tr>
</tbody>
</table>

Table A.2: Import and export capabilities of meta-modeling tools – *continued from previous page*
B Transformation Rules

B.1 ARIS-EMF Bridge

Listing B.1 shows the M2-transformation and Listing B.2 shows the M1-transformation of the ARIS-EMF bridge. Both transformations are expressed in ETL and are explained in detail in Section 6.2. The presented transformations are not directly executable. In fact, the purpose is the description of the transformation logic.

M2-Transformation

```java
rule Filter transform f: Aris ! Filter to p: Emf ! EPackage {
  p.name := f.name ;
  p.eClassifiers := f.objectTypes.equivalent();
  p.eClassifiers := f.connectionTypes.equivalent();
  p.eSubpackages := f.modelTypes.equivalent();
}

rule ObjectType transform o: Aris ! ObjectType to c: Emf ! EClass {
  c.name := o.name ;
  c.eAttributes := o.attributes.equivalent();
}

rule ConnectionType transform o: Aris ! ConnectionType to c: Emf ! EClass {
  c.name := o.name
  c.eAttributes := o.attributes.equivalent();
}

rule ModelType transform m: ModelType to p: EPackage {
  p.name := m.name ;
  var c := new Emf ! EClass ;
  c.name := m.name;
  c.eAttributes := m.attributes.equivalent();
  p.eClassifiers.addAll(m.symbols.equivalent());
}

rule Symbol transform s: Aris ! Symbol to c: Emf ! EClass {
  c.name := s.name
}

rule Attribute transform a: Aris ! Attribute to b: Emf ! EAttribute {
  b.name := a.name;
  b.eType := a.datatype;
}
```

Listing B.1: ARIS M2-transformation rules
M1-Transformation

```java
rule Model transform m: Aris!Model to e: Emf!Model {
  e.eClass := filterPackage.eClassifiers.
  selectOne(i|i.name=m.type.name);
  e.objectOccurrences := m.objectOccurences.equiv();
  e.connectionOccurances := m.connectionOccurences.equiv();
  for (a: Aris!Attribut in m.attributes) {
    e.eSet(e.getEStructuralFeature(a.type.name), a.value);
  }
}

rule ObjectOccurance
transform o: Aris!ObjectOccurance to e: Emf!ObjectOccurance {
  e.eClass := filterPackage.eClassifiers.
  selectOne(i|i.name = o.type.name);
  e.objectDefinition := o.objectDefinition.equiv();
}

rule ObjectDefinition
transform o: Aris!ObjectDefinition to e: Emf!ObjectDefinition {
  e.eClass := filterPackage.eClassifiers.
  selectOne(i|i.name = o.type.name);
  e.assignments := o.assignments.equiv();
  for (a: Aris!Attribut in o.attributes) {
    e.eSet(e.getEStructuralFeature(a.type.name), a.value);
  }
}

rule ConnectionOccurance
transform o: Aris!ConnectionOccurance to e: Emf!ConnectionOccurance {
  e.connectionDefinition := o.connectionDefinition.equivalence();
  e.source := o.source.equivalent();
  e.target := o.target.equivalent();
}

rule ConnectionDefinition
transform c: Aris!ConnectionDefinition to e: Emf!ConnectionDefinition {
  e.eClass := filterPackage.eClassifiers.
  selectOne(i|i.name = c.type.name);
  for (a: Aris!Attribut in c.attributes) {
    e.eSet(e.getEStructuralFeature(a.type.name), a.value);
  }
  e.source := c.source.equivalent();
  e.target := c.target.equivalent();
}
```

Listing B.2: ARIS M1-transformation rules
B.2 MetaEdit-EMF Bridge

Listing B.3 shows the M2-transformation and Listing B.4 shows the M1-transformation of the MetaEdit-EMF bridge. Both transformations are expressed in ETL and are explained in detail in Section 6.3. The presented transformations are not directly executable. In fact, the purpose is the description of the transformation logic.

**M2-Transformation**

```java
rule ObjectType
transform o: MetaEdit!ObjectType to c: Emf!EClass {
    c. name := o. typeName;
    c. eStructuralFeatures := o. properties. equivalent();
    c. eSupertypes := o. supertypes. equivalent();
}

rule RelationshipType
transform r: MetaEdit!RelationshipType to c: Emf!EClass {
    c. name := r. typeName;
    c. eStructuralFeatures := r. properties. equivalent();
    c. eSupertypes := r. supertypes. equivalent();
}

rule RoleType
transform r: MetaEdit!RoleType to c: Emf!EClass {
    c. name := r. typeName;
    c. eStructuralFeatures := r. properties. equivalent();
    c. eSupertypes := r. supertypes. equivalent();
}

rule GraphType
transform g: MetaEdit!GraphType to c: Emf!EClass {
    c. name := g. typeName;
    c. eStructuralFeatures := g. properties. equivalent();
    c. eSupertypes := g. supertypes. equivalent();
}

rule Property2Attribute
transform p: MetaEdit!Property to a: Emf!EAttribute {
    guard : p. type = 'SimpleType'
    a. name := p. typeName; a. eType := p. datatype;
    if (p. multivalue) a. upperBound(-1);
}

rule Property2Reference
transform p: MetaEdit!Property to r: Emf!ERefERENCE {
    guard : p. type = 'ElementType'
    r. name := p. typeName; r. eType := p. datatype;
    if (p. multivalue) a. upperBound(-1);
}
```

Listing B.3: MetaEdit M2-transformation rules
M1-Transformation

```plaintext
1 rule Graph
2   transform g:MetaEdit!MEOp to e:Emf!Graph {
3     guard : g.type = 'Graph'
4
5     e.eClass := gopprrPackage.eClassifiers
6     .selectOne(i|i.name = g.typeName);
7
8     e.objects := g.objectSet.equivalent();
9     e.relationship := g.relationshipSet.equivalent();
10    e.roles := g.roleSet.equivalent();
11    e.diagrams := g.diagrams.equivalent();
12
13    for(p:MetaEdit!Property in g.properties) {
14        e.eSet(e.getEStructuralFeature(p.typeName), p.value);
15    }
16 }
17
18 rule Object
19   transform o:MetaEdit!MEOp to e:Emf!Object {
20      guard : o.type = 'Object'
21
22      e.eClass := gopprrPackage.eClassifiers
23      .selectOne(i|i.name = o.typeName);
24
25      for(p:MetaEdit!Property in o.properties) {
26          e.eSet(e.getEStructuralFeature(p.typeName), p.value);
27      }
28
29      e.explosion := o.explodeGraph.equivalent();
30      e.decomposition := o.decompGraph.equivalent();
31  }
32
33 rule Relationship
34   transform r:MetaEdit!MEOp to e:Emf!Relationship {
35      guard : r.type = 'Relationship'
36
37      e.eClass := gopprrPackage.eClassifiers
38      .selectOne(i|i.name = r.typeName);
39
40      for(p:MetaEdit!Property in o.properties) {
41          e.eSet(e.getEStructuralFeature(p.typeName), p.value);
42      }
43
44      e.roles := r.rolesForRelationship.equivalent();
45      e.explosion := r.explodeGraph.equivalent();
46  }
47
48 rule Role
49   transform r:MetaEdit!MEOp to e:Emf!Role {
50      guard : r.type = 'Role'
51
52      e.eClass := gopprrPackage.eClassifiers
53      .selectOne(i|i.name = r.typeName);
54```
for (p: MetaEdit!Property in r.properties) {
    e.eSet(e.getEStructuralFeature(p.typeName), p.value);
}

e.objects := r.objectsForRole.equivalent();

e.explosion := r.explodeGraph.equivalent();
}

rule Diagram

transform d: MetaEdit!MEGop to e: Emf!Diagram {
    guard : d.type = 'Diagram'
    for(s: MetaEdit!Symbol in d.symbols) {
        var symbol := new Emf!Symbol;
        symbol.position := s.position;
        symbol.object := s.object.equivalent();
        e.symbols.add(symbol);
    }
}

Listing B.4: MetaEdit M1-transformation rules
B.3 Visio-EMF Bridge

Listing B.5 shows the M2-transformation and Listing B.6 shows the M1-transformation of the MetaEdit-EMF bridge. Both transformations are expressed in ETL and are explained in detail in Section 6.4. The presented transformations are not directly executable. In fact, the purpose is the description of the transformation logic.

M2-Transformation

```java
rule Stencil
  transform s:Visio!Stencil to p:Emf!EPackage {
    p.name := s.name;
    p.eClassifiers := s.masters.equivalent();
  }

rule ObjectMaster
  transform m:Visio!Master to c:EClass {
    guard : !m.isConnection
    c.name := m.name;
    emfMaster.eSuperTypes.add(objectShapeEClass);
    c.eAttributes := m.properties.equivalent();
  }

rule ObjectMaster
  transform m:Visio!Master to c:EClass {
    guard : m.isConnection
    c.name := m.name;
    emfMaster.eSuperTypes.add(connectionShapeEClass);
    c.eAttributes := m.properties.equivalent();
  }

rule Property
  transform p:Visio!Property to a:Emf!EAttribute {
    a.name := p.name;
    a.type := p.type;
    if(not p.value.isDefined())
      a.value := p.value;
  }
```

Listing B.5: Visio M2-transformation rules
M1-Transformation

```java
rule Page
  transform p:Visio!Page to e:Emf!Page {
    e.name := p.name;
    e.shapes := p.shapes.equivalent();
  }
rule ObjectShape
  transform s:Visio!Shape to e:Emf!SimpleShape {
    guard : not s.isConnectionShape
    e.eClass := stencilPackage.eClassifiers
    .selectOne(i| i.name = r.master.name);
    e.text := s.text;
    for (p: Visio!Property in s.properties) {
      e.eSet(e.getEStructuralFeature(p.name), p.value);
    }
  }
rule ConnectionShape
  transform s:Visio!Shape to e:Emf!ConnectionShape {
    guard : s.isConnectionShape
    e.eClass := stencilPackage.eClassifiers
    .selectOne(i| i.name = r.master.name);
    e.text := s.text;
    for (p: Visio!Property in s.properties) {
      e.eSet(e.getEStructuralFeature(p.name), p.value);
    }
    e.sourceShape := s.beginShape.equivalent();
    e.targetShape := s.endShape.equivalent();
  }
```

Listing B.6: Visio M1-transformation rules
C Meta-Model Alignment

This section refers to the combination of the MetaEdit-EMF and Visio-EMF bridge. The combination of both bridges is described in more detail in Section 6.5. The following listings define the transformation between MetaEdit-EMF and Visio-EMF models and vice versa.

C.1 MetaEdit+ to Visio

The transformation from MetaEdit-EMF models into Visio-EMF models consists of two parts. The first part is denoted as meta-model independent (MMI) transformation. The MMI-transformation includes rules for a generic mapping of MetaEdit-EMF models into Visio-EMF models. This transformation is presented in Listing C.1. The first rule Project2Document creates a Visio document for each MetaEdit+ project. The second rule Graph2Page creates for each graph in a project a corresponding page in a Visio document. The third rule Shape creates a Visio shape for each MetaEdit+ object. The operation setPosition is responsible for transformation of coordinates.

```
1 rule Project2Document
2   transform project : MetaEdit!Me_Project
3     to document : Visio!EVisioDocument {
4       document.visioPages := project.me_containGraphs.equivalent();
5     }
6
7 @abstract
8 rule Graph2Page
9   transform graph : MetaEdit!Me_Graph
10      to page : Visio!EVisioPage {
11         page.visioContainedShapes :=
12             graph.me_referenceObjects.equivalent();
13             page.visioContainedShapes.addAll(graph.  
14                me_referenceRelationships.  
15                   equivalent());
16     }
17
18 @abstract
19 rule Shape
20   transform object : MetaEdit!Me_Object
21      to shape : Visio!EVisioShape {
22       var position := MetaEdit!Me_Symbol.allInstances()  
23          .select(e | e.me_refObject = object).first().me_position;
24```
shape.setPosition(position);
}

operation Visio.EVisioShape setPosition(position : String) : Integer {
    var x := position.split(",").first();
    var y := position.split(",").last();
    var shapeSheet := new Visio.EVisioShapeSheet;
    var section := new Visio.EVisioSection;
    section.visioName := 'visSectionObject';
    var row := new Visio.EVisioRow;
    row.visioName := '1';
    var xCell := new Visio.EVisioCell;
    xCell.visioName := 'PinX';
    xCell.visioValue := x;
    var yCell := new Visio.EVisioCell;
    yCell.visioName := 'PinY';
    yCell.visioValue := y;
    self.visioShapeSheet := shapeSheet;
    shapeSheet.visioSections.add(section);
    section.visioRows.add(row);
    row.visioCells.add(xCell);
    row.visioCells.add(yCell);
}

post CoordinateTransformation {
    for (pinY : Visio.EVisioCell in Visio.EVisioCell.allInstances.
        select(e | e.visioName = 'PinY'))
    {
        var y : Real := pinY.visioValue.asReal();
        y := (1150-y)/100;
        pinY.visioValue := y+'';
    }
    for (pinX : Visio.EVisioCell in Visio.EVisioCell.allInstances.
        select(e | e.visioName = 'PinX'))
    {
        var x : Real := pinX.visioValue.asReal();
        x := x/100;
        pinX.visioValue := x+'';
    }
}

Listing C.1: MMI-transformation from MetaEdit+ to Visio
The second part is denoted as meta-model dependent (MMD) transformation. The MMD-transformation extends the MMI-transformation and depends on concrete MetaEdit-EMF and Visio-EMF meta-models. In this case, the transformation is specific for the EPC language. The following Listing C.2 shows the mapping between EPC models from MetaEdit+ to Visio. The MMD-transformation can be automatically created by a generator. The input of this generator is a mapping description between participating meta-models.

The first rule \textit{MetaEditEpc2VisioEpc} in line 1 extends the \textit{Graph2Page} rule from the MMI transformation. This rule maps a MetaEdit+ EPC graph to a Visio page. The next rule in line 9 \textit{MetaEditEvent2VisioEvent} maps MetaEdit+ event objects into Visio event shapes. All other rules are similar except the last rule \textit{MetaEditArc2VisioConnection} in line 40. This rule maps a MetaEdit EPC arc into a Visio connection master.
rule MetaEditArc2VisioConnection
transform relation : MetaEdit!Arc_3395083800
to shape : Visio!Dynamischer_Verbinder {
  for (role : MetaEdit!Me_Role in relation.me_role) {
    if (role.isTypeOf(MetaEdit!To_3395083810))
      shape.visioTargetShape := role.me_object.equivalent();
    if (role.isTypeOf(MetaEdit!From_3395083804))
      shape.visioSourceShape := role.me_object.equivalent();
  }
}

Listing C.2: MMD-transformation from MetaEdit+ to Visio

C.2 Visio to MetaEdit+

This transformation is the counterpart of the previous transformation. This transformation converts Visio-EMF models into MetaEdit-EMF models. The transformation also comprises two parts. The first part offers rules independent of a specific meta-model. The MMI-transformation is presented in the following Listing C.3. The first rule Document2Project in line 1 creates a MetaEdit+ project for each Visio document. The second rule Page2Graph in line 12 creates a MetaEdit+ graph for each Visio page. The two operations are responsible for the transformation of coordinates.

rule Document2Project
transform document : Visio!EVisioDocument
to project : MetaEdit!Me_Project {
  project.me_containGraphs := document.visioPages.equivalent();
  project.me_containObjects := MetaEdit!Me_Object.allInstances();
  project.me_containRelationships := MetaEdit!Me_Relationship.
                                 allInstances();
  project.me_containRoles := MetaEdit!Me_Role.allInstances();
}

@abstract
rule Page2Graph
transform page : Visio!EVisioPage
to graph : MetaEdit!Me_Graph {
  var diagram := new MetaEdit!Me_Diagram;
  graph.me_containDiagrams.add(diagram);
  var simpleShapes := page.visioContainedShapes->select(e | e not e.isKindOf(Visio!EVisioConnectionShape));
  graph.me_referenceObjects := simpleShapes.equivalent();
  for (shape : Visio!EVisioShape in simpleShapes) {
    var symbol := new MetaEdit!Me_Symbol;
C.2 Visio to MetaEdit+

```plaintext
23 diagram.me_containSymbols.add(symbol);
24 symbol.me_refObject := shape.equivalent();
25 symbol.me_position := shape.getPinX()+','+shape.getPinY();
26 }
27
28 var conShapes := page.visioContainedShapes-&gt;select(e | e.
isKindOf(Visio!EVisioConnectionShape));
29 graph.me_referenceRelationships := conShapes.equivalent().
   select(e | e.isKindOf(MetaEdit!Me_Relationship));
30 graph.me_referenceRoles := conShapes.equivalent().select(e | e .isKindOf(MetaEdit!Me_Role));
31 }
32
33 operation Visio!EVisioShape getPinX() : Integer {
34   return self.visioShapeSheet
35     .visioSections.select(e | e.visioName = 'visSectionObject').
36       first()
37     .visioRows.select(e | e.visioName = '1').first()
38     .visioCells.select(e | e.visioName = 'PinX').first()
39     .visioValue.asReal();
40 }
41
42 operation Visio!EVisioShape getPinY() : Integer {
43   return self.visioShapeSheet
44     .visioSections.select(e | e.visioName = 'visSectionObject').
45       first()
46     .visioRows.select(e | e.visioName = '1').first()
47     .visioCells.select(e | e.visioName = 'PinY').first()
48     .visioValue.asReal();
49 }
50
51 post CoordinateTransformation {
52   var ymax : Real := MetaEdit!Me_Symbol.allInstances.sortBy(e|e.
53     me_position.split(',').last()).
54       last().me_position.split(',').last().asReal();
55   for (symbol : MetaEdit!Me_Symbol in MetaEdit!Me_Symbol.
56     allInstances) {
57     var x : Real := symbol.me_position.split(',').first().asReal();
58     var y : Real := symbol.me_position.split(',').last().asReal();
59     x := x*100;
60     y := (ymax*120+20)-y*100;
61     symbol.me_position := x.asInteger() + ',' + y.asInteger();
62   }
63 }
```

Listing C.3: MMI-transformation from Visio to MetaEdit+
The second part describes the transformation of EPC models from Visio-EMF to MetaEdit-EMF. Listing C.4 shows the MMD-transformation. The rules define a mapping between EPC language elements. The first rule `VisioEpc2MetaEditEpc` in line 1 `VisioEpc2MetaEditEpc` creates an EPC graph for each Visio page. The second rule `VisioEvent2MetaEditEvent` in line 8 creates a MetaEdit+ object for each Visio event. The last rule `VisioConnection2MetaEditArc` in line 32 is responsible for the transformation of relationships.

```java

rule VisioEpc2MetaEditEpc
    transform page : Visio!EVisioPage
to graph : MetaEdit!Event_Driven_Process_Chain_3395083925
extends Page2Graph {
    graph.Name := page.visioName;
}

rule VisioEvent2MetaEditEvent
    transform shape : Visio!Ereignis
to object : MetaEdit!Event_3395083771 {
    object.Name := shape.visioText;
}

rule VisioFunction2MetaEditFunction
    transform shape : Visio!Funktion
to object : MetaEdit!Function_3395083784 {
    object.Name := shape.visioText;
}

rule VisioXOR2MetaEditXOR
    transform shape : Visio!XOR
to object : MetaEdit!XOR_3395083733 {}

rule VisioOR2MetaEditOR
    transform shape : Visio!OR
to object : MetaEdit!OR_3395083742 {}

rule VisioAND2MetaEditAND
    transform shape : Visio!AND
to object : MetaEdit!AND_3395083748 {}

rule VisioConnection2MetaEditArc
    transform shape : Visio!Dynamischer_Verbinder
to relation : MetaEdit!Arc_3395083800,
    fromRole : MetaEdit!From_3395083804,
    toRole : MetaEdit!To_3395083810 {
    relation.me_role.add(fromRole);
    relation.me_role.add(toRole);
    fromRole.me_object := shape.visioSourceShape.equivalent();
    toRole.me_object := shape.visioTargetShape.equivalent();
}
```

Listing C.4: MMD-transformation from Visio to MetaEdit+
D Research Work

This section enumerates publications and prototypes created during the research for this thesis. The publications are written by the author of this thesis and co-authors. The publications are structured into different parts. The first part comprises publications relating to the M3B approach itself. The second part concentrates on the aspect of meta-modeling. The third part presents applications of the M3B approach. The publications are part of the research design and contribute to the communication in the research community. This communication helped to improve the research work. The developed prototypes implement the M3B approach and served as use cases and evaluation objects.

D.1 Publications

M3-Level-Based Bridges

- [Dimitrieski et al. 2015]: Reuse of Rules in a Mapping-Based Integration Tool
- [Kern et al. 2014]: Mapping-Based Exchange of Models Between Meta-Modeling Tools
- [Kern 2014]: Study of Interoperability between Meta-Modeling Tools
- [Kern 2012]: Interoperabilität mittels M3-Level-basierter Transformationen
- [Hilner and Kern 2010]: Bridging Microsoft Oslo and Eclipse EMF
- [Kern and Kühne 2009]: Integration of Microsoft Visio and Eclipse Modeling Framework Using M3-Level-Based Bridges
- [Kern et al. 2009a]: Ansatz zur ganzheitlichen Erzeugung und Verarbeitung von Dienstleistungsmodellen
- [Kern 2008a]: Interchange of (Meta)Models between MetaEdit+ and Eclipse EMF using M3-Level-Based Bridges
- [Kern 2008b]: Modellaustausch zwischen ARIS und Eclipse EMF durch Verwendung einer M3-Level-basierten Brücke
- [Kern and Kühne 2007a]: Model Interchange between ARIS and Eclipse EMF

Meta-Modeling

- [Kern et al. 2011]: Towards a Comparative Analysis of Meta-metamodels
- [Kern 2008b]: Metamodellierung aus Sicht von ARIS
Application

- [Kern et al. 2015a]: Intelligent and Self-Adapting Integration between Machines and Information Systems
- [Kern et al. 2015b]: A Mapping-Based Framework for the Integration of Machine Data and Information Systems
- [Kühne et al. 2010]: Business process modeling with continuous validation
- [Kern et al. 2009b]: Modellinteroperabilität zwischen Microsoft Visio und Eclipse EMF als Mittel zur modellgetriebenen Integration

D.2 Prototypes

Prototype 1: ARIS-EMF Bridge

This bridge allows the migration of meta-models and models from ARIS to EMF. The bridge was the first implementation of the M3B approach and is presented in more detail in Section 6.2.

Prototype 2: MetaEdit-EMF Bridge

This bridge allows the migration of meta-models from MetaEdit+ to EMF and the migration of models in both directions. The MetaEdit-EMF bridge is described in Section 6.3. The implementation is publicly available on the Internet\(^1\).

Prototype 3: Visio-EMF Bridge

The third prototype enables the migration of Visio stencils to EMF meta-models and the migration of corresponding models from Visio to EMF and vice versa. The bridge is described in Section 6.4. The source code is available as an open source project at SourceForge\(^2\). The bridges was download more than 2.400 times in five years.

Prototype 4: AnyMap

This prototype implements the combination concept of M3Bs described in Section 4.4. The tool uses M3Bs to import and export models and meta-models from different meta-modeling tools. AnyMap offers a mapping language enabling the definition of mappings between different meta-models. Based on this mapping, AnyMap uses different generators to create executable transformation code. The development of this tool is a joint project between the University of Leipzig and the University of Novi Sad.

\(^1\)http://www.informatik.uni-leipzig.de/~kern/metaedit.emf.bridge_1.1.0.jar
\(^2\)http://sourceforge.net/projects/visioemfbridge/
Bibliography

Agrawal et al. 2003
AGRAWAL, Aditya ; KARSAI, Gabor ; SHI, Feng: Graph Transformations on Domain-Specific Models. (2003). – URL http://w3.isis.vanderbilt.edu/publications/archive/Agrawal_A_11_0_2003_Graph_Tran.pdf

van Amstel et al. 2008

van Amstel and van den Brand 2011

Atkinson and Gerbig 2012

Atkinson and Kühne 2001

Atkinson and Kühne 2003
Barry and Stanienda 1998

Bauer et al. 2015

Baumann et al. 2000

von Bertalanffy 1976

Bézivin 2006

Biafore 2007

Boehm et al. 1978

Boehm et al. 1976

Brandes et al. 2002
Bibliography

Chen 2005

Chen 1976

Cook et al. 2007

Czarnecki and Eisenecker 2000

Czarnecki and Mens 2003

Date 2008

Demathieu et al. 2005
Demathieu, Sebastian; Griffin, Catherine; Sendall, Shane: Model Transformation with the IBM Model Transformation Framework. (2005), May. – URL http://www.ibm.com/developerworks/rational/library/05/503_sebas/

Dimitreski et al. 2015

Efftinge et al. 2008
Efftinge, Sven; Friese, Peter; Haase, Arno; Hübner, Dennis; Kadura, Clemens; Kolb, Bernd; Köhnlein, Jan; Moroff, Dieter; Thoms,
Karsten; Völter, Markus; Schönbach, Patrick; Eysoldt, Moritz; Hübner, Dennis; Reinish, Steven: openArchitectureWare User Guide Version 4.3.1. (2008)

Ehrig et al. 2006

Englebert and Magusiak 2013

Erdweg et al. 2015

Ermel et al. 1999

Favre 2005

Fill et al. 2012

Fill et al. 2013
FILL, Hans-Georg; REDMOND, Timothy; KARAGIANNIS, Dimitris: Formalizing Meta Models with FDMM: The ADOxx Case. In: CORDEIRO, Jose (Ed.);

Fowler 2003

Frank 2011

Frank 2014

Frankel 2003

van Gigch 1991

Gold-Bernstein and Ruh 2004

Gove 1993

Greenfield et al. 2004

Greiffenberg 2003
Gronback 2009  

Halstead 1977  

Harel and Rumpe 2004  
Harel, David; Rumpe, Bernhard: Meaningful Modeling: What’s the Semantics of ”Semantics”? In: *Computer* 37 (2004), October, No. 10, p. 64–72. – ISSN 0018-9162

Hein et al. 2009  

Hevner et al. 2004  

Hillner and Kern 2010  

Hopcroft et al. 2006  

IEEE 1990  

IEEE 1998  

ISIS 2006  
Jawin 2005


Jouault 2006


Jouault and Kurtev 2006


Junginger et al. 2000


Kammermeier et al. 2011


Kan 2002


Karagiannis and Kühn 2002


Karsai et al. 2003

Kelly 1997

Kelly and Tolvanen 2008

Kern 2007a

Kern 2007b

Kern 2008a

Kern 2008b

Kern 2012

Kern 2014
KERN, Heiko: Study of Interoperability between Meta-Modeling Tools. In:

Kern et al. 2009a

Kern et al. 2011

Kern et al. 2009b

Kern and Kühne 2007a

Kern and Kühne 2007b

Kern and Kühne 2009
KERN, Heiko ; KÜHNE, Stefan: Integration of Microsoft Visio and Eclipse Modeling Framework Using M3-Level-Based Bridges. In: HEIN, Christian (Ed.) ; RITTER, Tom (Ed.) ; WAGNER, Michael (Ed.): *Proceedings of Second Workshop on Model-Driven Tool and Process Integration (MDTPI) at Fifth*

Kern et al. 2006

Kern et al. 2015a

Kern et al. 2014

Kern et al. 2015b

Kirchner and Jung 2007

Kolovos 2008

Kolovos et al. 2015

Kühne et al. 2010
Kühne, Stefan; Kern, Heiko; Gruhn, Volker; Laue, Ralf: Business process
modeling with continuous validation. In: *Journal of Software Maintenance and Evolution: Research and Practice* 22 (2010), No. 6-7, p. 547–566. – URL http://dx.doi.org/10.1002/smr.517. – ISSN 1532-0618

Kühne 2006

Kurtev et al. 2002
KURTEV, Ivan ; BÉZIVIN, Jean ; AKSIT, Mehmet: Technological Spaces: an Initial Appraisal. In: *Proceedings of International Federated Conferences (OTM’02), Industry Track*, 2002

Lamo et al. 2012

de Lara and Guerra 2010

Ledeczi et al. 2001

Loos and Fettke 2007

Ludwig and Salger 2006

Lundell et al. 2006
LUNDELL, Björn ; LINGS, Brian ; PERSSON, Anna ; MATTSSON, Anders: UML Model Interchange in Heterogeneous Tool Environments: An Analysis of Adoptions of XMI 2. In: NIERSTRASZ, Oscar (Ed.) ; WHITTLE, Jon (Ed.) ; HAREL, David (Ed.) ; REGGIO, Gianna (Ed.): *Model Driven Engineering Languages*
Marwick et al. 2014

McCall et al. 1977

McLaughlin and Edelson 2006

Mellor et al. 2004

Mens and van Gorp 2006

Mens et al. 2006
Mens, Tom; Gorp, Pieter van; Varró, Déniel; Karsai, Gabor: Applying a Model Transformation Taxonomy to Graph Transformation Technology. In: Electronic Notes in Theoretical Computer Science 152 (2006), March, p. 143–159. – ISSN 1571-0661

Mohagheghi and Dehlen 2007

Molina et al. 2007
Molina, Arturo; Panetto, Hervé; Chen, David; Whitman, Lawrence; Chapurlat, Vincent; Vernadat, François: Enterprise Integration and Networking: Challenges and Trends. In: Studies in Informatics and Control 16 (2007), No. 4, p. 353–368
OASIS 2007

OMG 2003

OMG 2011

OMG 2014a

OMG 2014b

OMG 2015a

OMG 2015b

OMG 2015c

Popma 2004

Prechtl and Burkard 1999

Royce 1987
Sandkuhl et al. 2014

Schmidt 2006

Stachowiak 1973

Stahl and Völter 2006

Stefan et al. 2013

Stein et al. 2008

Steinberg et al. 2009

Strahringer 1996
Bibliography

Sztipanovits and Karsai 1997

Thomas 2002

Tisi et al. 2009

Tolvanen 1998

Vanhooff et al. 2006

Visser 2001

W3C 2008
W3C: Extensible Markup Language (XML) 1.0 (Fifth Edition). (2008), November. – URL http://www.w3.org/TR/xml/

W3C 2011

Wagelaar et al. 2010
Wagelaar, Dennis; Van Der Straeten, Ragnhild; Deridder, Dirk: Module superimposition: a composition technique for rule-based model transforma-
– ISSN 1619-1366

**Wasserman 1990**


**Weske 2012**


**WFMC 2012**


**Wicks 2006**


**Wimmer 2005**


**Winter et al. 2002**

Scientific Background

Research Projects

10.2003 – 12.2006  Integration Engineering–Entwicklung eines Vorgehensmodells zur Umsetzung kooperativer Geschäftsprozesse auf eine integrierende, internet-basierte IT-Struktur; BMBF project; ID: 01ISC18; project member

10.2005 – 09.2008  Orchestrierung und Validierung integrierter Anwendungssysteme (OrViA); BMBF project; ID: 01ISE10; project member

01.2008 – 07.2010  Automatisierte Anpassung, Integration, Evolution und Migration von Full-Service-Anwendungen im E-Commerce (autoFuSA); BMBF project; ID: 01IS08010; project lead

11.2008 – 10.2010  Advanced Model Repository (AMOR); BMBF project; ID: 01IS08038; project member

12.2010 – 03.2013  Modellierung, Simulation und Prognostik von E- Commerce-Wirkungsketten (SimProgno); BMBF project; ID: 01IS10042; project lead

03.2013 – 10.2015  Entwicklung einer generischen und hochmodularen Hardwarelösung für das dezentrale Monitoring von Maschinen und Anlagen auf Basis eines Embedded Enterprise Service Bus (ZentralMonitor); BMWI-Projekt; ID 16KN015621; project member

02.2014 – 12.2015  Discovering Effective Methods and Architectures for Integration of Modeling Spaces with Applications in Various Problem Domains; DAAD project; ID: 57059661; project lead

03.2014 – 10.2014  Framework zur Wiederverwendung geschäftsojektorientierter Transformationen im Bereich der Anwendungsintegration (WiGoTa); SAB project; ID: 100188035/2926; project lead
05.2014 – 01.2016 Selbstlernender Automat zur intelligenten Anbindung von Maschinen an betriebliche Informationssysteme (SAAMI); BMWI project; ID:KF2495396BZ3; project lead

01.2015 – 12.2017 Self-Adapting Interface Framework for the Integration of Machines and Information Systems; DAAD project; ID:57215093; project lead

03.2016 – 02.2018 Ein adaptives Deeskalationsmanagementsystem für die Logistik (ADILO); SAB project; ID:3000640391; project lead

Participation in Committees

2009 IEEE Software: Special Issue on Domain-Specific Languages and Modeling; reviewer
Workshop Integration Engineering; member of organization committee

2012 12th Workshop on Domain-Specific Modeling at OOPSLA/SPLASH 2012; member of program committee
Enterprise Modelling and Information Systems Architectures – An International Journal; reviewer
First Workshop on Graphical Modeling Language Development at ECMFA 2012; member of organization and program committee

2013 Second Workshop on Graphical Modeling Language Development at ECMFA 2013; member of organization and program committee
Studierendenkonferenz Informatik (SKILL); member of program committee

2014 Electronic Commerce Research and Applications, Journal; reviewer
14th Workshop on Domain-Specific Modeling at SPLASH 2014; member of program committee
Studierendenkonferenz Informatik (SKILL); member of program committee
Computer Science and Information Systems (ComSIS) Journal; reviewer

2015 Computer Languages, Systems and Structures – An International Journal; reviewer
Studierendenkonferenz Informatik (SKILL); member of program committee
12. Internationale Tagung Wirtschaftsinformatik; reviewer

2016 Computer Science and Information Systems (ComSIS) Journal; reviewer

**Supervised Theses**


**Selected Talks**

2007 Model Interchange between ARIS and Eclipse EMF; 7th Workshop on Domain-Specific Modeling at ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA); Montréal, Canada, 22. October 2007


Interchange of (Meta)Models between MetaEdit+ and Eclipse EMF using M3-Level-Based Bridges; 8th Workshop on Domain-Specific Modeling at ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA); Nashville, Tennessee, USA, 19. October 2008

2009  Integration of Microsoft Visio and Eclipse Modeling Framework using M3-Level-Based Bridges; Second Workshop on Model-Driven Tool and Process Integration (MDTPi) at Fifth European Conference on Model-Driven Architecture Foundations and Applications (ECMFA); Enschede, The Netherlands, 24. July 2009

2011  Metamodellierung und Modelltransformation; invited talk at Chair of Information Systems and Enterprise Modelling, Institute for Computer Science and Business Information Systems, University of Duisburg-Essen, Essen, Germany, 25. May 2011

Towards a Comparative Analysis of Meta-Metamodels; 11th Workshop in Domain-Specific Modeling at ACM SIGPLAN Conference on Systems, Programming, Languages and Applications (SPLASH); Portland, Oregon, USA, 23. October 2011

2013  Modellierung, Simulation und Prognostik von E-Commerce-Wirkungsketten; CeBIT; Hannover, Germany 05.–09. March 2013

2014  Model Transformation between Meta-Modeling Environments by using M3-Level-Based Bridges, invited talk at Chair for Applied Computer Science, Computing and Control Department, Faculty of Technical Sciences, University of Novi Sad; Novi Sad, Serbia, 24. April 2014

Study of Interoperability between Meta-Modeling Tools; Third Workshop on Model Driven Approaches in System Development (MDASD) at Federated Conference on Computer Science and Information Systems (FedCSIS); Warsaw, Poland, 08. September 2014

Mapping-Based Exchange of Models Between Meta-Modeling Tools; 14th Workshop in Domain-Specific Modeling at ACM SIGPLAN Conference on Systems, Programming, Languages and Applications: Software for Humanity (SPLASH); Portland, Oregon, USA, 21. October 2014
A Mapping-Based Framework for the Integration of Machine Data and Information Systems; 8th IADIS International Conference on Information Systems 2015; Funchal, Madeira, Portugal, 16 March 2015

Discovering Effective Methods and Architectures for Integration of Modeling Spaces with Applications in Various Problem Domains, invited talk at Chair for Applied Computer Science, Computing and Control Department, Faculty of Technical Sciences, University of Novi Sad; Novi Sad, Serbia, 11 June 2015

Selbstlernender Automat zur intelligenten Anbindung von Maschinen an betriebliche Informationssysteme; SPS IPC Drives; Hannover, Germany 24.–26. November 2015

Publications

Journals


KÜHNE, Stefan; KERN, Heiko; GRUHN, Volker; LAUE, Ralf: Business process modeling with continuous validation. In: Journal of Software Maintenance and Evolution: Research and Practice 22 (2010), No. 6-7, p. 547–566. – ISSN 1532-0618


Conferences and Workshops


Book Chapters


Books


KERN, Heiko (Ed.) ; KÜHNE, Stefan (Ed.): Integration betrieblicher Informationssysteme und modellgetriebene Entwicklung. Eigenverlag Leipziger Informatik-Verbund (LIV), April 2012. – ISBN 978-3-941608-17-7

Miscellaneous

HILLNER, Stanley ; KERN, Heiko: Bridging Microsoft Oslo and Eclipse EMF / University Leipzig. 2010. – Technical report

Bibliografische Daten

**Titel:** Model Interoperability between Meta-Modeling Environments by using M3-Level-Based Bridges

**Autor:** Heiko Kern

**Institution:** Universität Leipzig, Fakultät für Mathematik und Informatik

**Umfang:** 205 Seiten, 44 Abbildungen, 27 Tabellen, 4 Anhänge, 131 Literaturangaben