# Structural Graph-based Metamodel Matching

**Dissertation**

zur Erlangung des akademischen Grades Doktoringenieur (Dr.-Ing.)

vorgelegt an der
Technischen Universität Dresden
Fakultät Informatik

eingereicht von

**Dipl.-Inf. Konrad Voigt**
geboren am 21. Januar 1981 in Berlin

Gutachter:
Prof. Dr. rer. nat. habil. Uwe Aßmann (Technische Universität Dresden)
Prof. Dr. Jorge Cardoso (Universidade de Coimbra, PT)

Tag der Verteidigung: Dresden, den 2. November 2011

Dresden im Dezember 2011

## Abstract

Data integration has been, and still is, a challenge for applications processing multiple heterogeneous data sources. Across the domains of schemas, ontologies, and metamodels, this imposes the need for mapping specifications, i.e. the task of discovering semantic correspondences between elements. Support for the development of such mappings has been researched, producing matching systems that automatically propose mapping suggestions.

However, especially in the context of metamodel matching the result quality of state of the art matching techniques leaves room for improvement. Although the traditional approach of pair-wise element comparison works on smaller data sets, its quadratic complexity leads to poor runtime and memory performance and eventually to the inability to match, when applied on real-world data.

The work presented in this thesis seeks to address these shortcomings. Thereby, we take advantage of the graph structure of metamodels. Consequently, we derive a planar graph edit distance as metamodel similarity metric and mining-based matching to make use of redundant information. We also propose a planar graph-based partitioning to cope with large-scale matching. These techniques are then evaluated using real-world mappings from SAP business integration scenarios and the MDA community. The results demonstrate improvement in quality and managed runtime and memory consumption for large-scale metamodel matching.

# Acknowledgements

This dissertation was conducted at SAP Research Dresden directed by Dr. Gregor Hackenbroich. I am grateful to SAP Research for financing this work through a pre-doctoral position.

Further, I want to express my gratitude to my advisor Prof. Uwe Aßmann as well as to Prof. Jorge Cardoso and Prof. Alexander Schill who agreed to be my co-advisors. I thank my supervisor Uwe Aßmann for giving me the opportunity to work on this interesting and challenging topic and for all the advises he gave me and Prof. Alexander Schill for constructive comments on this work. I owe a great debt to Prof. Jorge Cardoso, whom I had the pleasure to work with. Jorge has been the kind of mentor to me who all young researchers should have.

I am grateful to Petko Ivanov, Thomas Heinze, Peter Mucha, and Philipp Simon. It was very motivating to discuss with them and the thesis benefited a lot from their contributions. A big thanks to you students, without you this thesis would have been impossible.

Further, I would like to thank the people from the TU Dresden and SAP Research who helped me with their input and discussions. Special thanks go to the ones commenting on my work: Eldad Louw for his perfect English and joy, Dr. Kay Kadner for his support in the TEXO project and magic moments, Eric Peukert for valuable discussions and expert knowledge exchange, Birgit Grammel for reminding me of myself and sharing some of the PhD agonies, Dr. Andreas Rummler for the mentoring, Dr. Roger Kilian-Kehr for critical thoughts, Dr. Karin Fetzer for comments on short notice, Arne for physical and psychological work-outs, and Daniel Michulke for regular lunch-meetings. I am grateful to Dr. Gregor Hackenbroich for his support, his valuable and precise comments, and for allowing me to continue my work at SAP Research, and I would also like to thank Annette Fiebig who supported me not only in administrative issues.

I also would like to thank my friends for tolerating and enduring my absence and lust for work. Thank you for still knowing me. Finally, I want to thank my family for encouragement and enjoyable moments.

And thank you Karen not only for reading countless versions of my thesis and commenting each of them thoroughly but also thank you for all your love and trust in me. Thank you.

# Contents

**1 Introduction** **1**
1.1 Quality Problem in Matching . . . . . . . . . . . . . . . . . . . . 3
1.2 Scalability Problem in Matching . . . . . . . . . . . . . . . . . . . 4
1.3 Research Questions and Contributions . . . . . . . . . . . . . . 4
1.4 Thesis Outline . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

**2 Background** **9**
2.1 Metamodel Matching . . . . . . . . . . . . . . . . . . . . . . . . 9
2.1.1 Metamodel . . . . . . . . . . . . . . . . . . . . . . . . . 9
2.1.2 Matching . . . . . . . . . . . . . . . . . . . . . . . . . . 11
2.2 Graph Theory . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
2.2.1 Definitions . . . . . . . . . . . . . . . . . . . . . . . . . 19
2.2.2 Metamodel representation . . . . . . . . . . . . . . . . 20
2.2.3 Graph properties . . . . . . . . . . . . . . . . . . . . . 24
2.2.4 Graph matching . . . . . . . . . . . . . . . . . . . . . . 28
2.2.5 Graph mining . . . . . . . . . . . . . . . . . . . . . . . 30
2.2.6 Graph partitioning and clustering . . . . . . . . . . . . 31
2.3 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33

**3 Problem Analysis** **35**
3.1 Motivating Example . . . . . . . . . . . . . . . . . . . . . . . . 35
3.1.1 Retail scenario description . . . . . . . . . . . . . . . . 35
3.1.2 ERP and POS metamodels . . . . . . . . . . . . . . . . 37
3.1.3 Data integration problems . . . . . . . . . . . . . . . . 38
3.2 Problem Analysis . . . . . . . . . . . . . . . . . . . . . . . . . . 40
3.2.1 Problems and scope . . . . . . . . . . . . . . . . . . . . 40
3.2.2 Objectives . . . . . . . . . . . . . . . . . . . . . . . . . 44
3.2.3 Requirements . . . . . . . . . . . . . . . . . . . . . . . . 45
3.2.4 Approach . . . . . . . . . . . . . . . . . . . . . . . . . . 46
3.2.5 Research question . . . . . . . . . . . . . . . . . . . . . 48
3.3 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 48

**4 Related Work** **51**
4.1 Matching Systems . . . . . . . . . . . . . . . . . . . . . . . . . 51
4.1.1 Schema matching . . . . . . . . . . . . . . . . . . . . . . 52
4.1.2 Ontology matching . . . . . . . . . . . . . . . . . . . . . 53
4.1.3 Metamodel matching . . . . . . . . . . . . . . . . . . . . 54
4.2 Matching Quality . . . . . . . . . . . . . . . . . . . . . . . . . . 56
4.3 Matching Scalability . . . . . . . . . . . . . . . . . . . . . . . . 58
4.4 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 60

**5 Structural Graph Edit Distance and Graph Mining Matcher** **63**
5.1 Planar Graph Edit Distance Matcher . . . . . . . . . . . . . . 63
5.1.1 Analysis of graph matching algorithms . . . . . . . . . 64
5.1.2 Planar graph edit distance algorithm . . . . . . . . . . 67
5.1.3 Example calculation . . . . . . . . . . . . . . . . . . . . 70
5.1.4 Improvement by k-max degree partial seed matches . 72
5.2 Graph Mining Matcher . . . . . . . . . . . . . . . . . . . . . . . 73
5.2.1 Analysis of graph mining algorithms . . . . . . . . . . 75
5.2.2 Graph model for mining based matching . . . . . . . . 78
5.2.3 Design pattern matcher . . . . . . . . . . . . . . . . . . 79
5.2.4 Redundancy matcher . . . . . . . . . . . . . . . . . . . . 84
5.3 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 89

**6 Planar Graph-based Partitioning for Large-scale Matching** **91**
6.1 Partition-based Matching . . . . . . . . . . . . . . . . . . . . . 91
6.2 Planar Graph-based Partitioning . . . . . . . . . . . . . . . . . 93
6.2.1 Analysis of graph partitioning algorithms . . . . . . . . 93
6.2.2 Planar Edge Separator based partitioning . . . . . . . 96
6.3 Assignment of Partitions for Matching . . . . . . . . . . . . . 102
6.3.1 Partition similarity . . . . . . . . . . . . . . . . . . . . . 103
6.3.2 Assignment algorithms . . . . . . . . . . . . . . . . . . . 104
6.3.3 Comparison . . . . . . . . . . . . . . . . . . . . . . . . . 109
6.4 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 110

**7 Evaluation** **113**
7.1 Evaluation strategy . . . . . . . . . . . . . . . . . . . . . . . . . 113
7.2 Evaluation framework: MatchBox . . . . . . . . . . . . . . . . 115
7.2.1 Processing steps and architecture . . . . . . . . . . . . 115
7.2.2 Matching techniques . . . . . . . . . . . . . . . . . . . . 116
7.2.3 Parameters and configuration . . . . . . . . . . . . . . . 118
7.3 Evaluation Data Sets . . . . . . . . . . . . . . . . . . . . . . . . 119
7.3.1 Data set metrics . . . . . . . . . . . . . . . . . . . . . . . 119
7.3.2 Enterprise service repository mappings . . . . . . . . . 121
7.3.3 ATL-zoo mappings . . . . . . . . . . . . . . . . . . . . . 126
7.3.4 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . 131

7.4 Evaluation Criteria . . . . . . . . . . . . . . . . . . . . . . . . . 132
7.5 Results for Graph-based Matching . . . . . . . . . . . . . . . . . 133
7.5.1 Graph edit distance results . . . . . . . . . . . . . . . . . 135
7.5.2 Graph mining results . . . . . . . . . . . . . . . . . . . . . 141
7.5.3 Discussion . . . . . . . . . . . . . . . . . . . . . . . . . . . 144
7.6 Results for Graph-based Partitioning . . . . . . . . . . . . . . . 144
7.6.1 Partition size . . . . . . . . . . . . . . . . . . . . . . . . . 145
7.6.2 Partition assignment . . . . . . . . . . . . . . . . . . . . . 147
7.6.3 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . 151
7.7 Discussion of Results . . . . . . . . . . . . . . . . . . . . . . . . 152
7.7.1 Applicability . . . . . . . . . . . . . . . . . . . . . . . . . 152
7.7.2 Limitations . . . . . . . . . . . . . . . . . . . . . . . . . . 153
7.8 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 154

**8 Conclusion** **157**
8.1 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 157
8.2 Conclusion and Contributions . . . . . . . . . . . . . . . . . . . 159
8.3 Recommendations for Data Model Development . . . . . . . . 161
8.4 Further Work . . . . . . . . . . . . . . . . . . . . . . . . . . . . 163

**A Evaluation Data Import** **167**
A.1 ATL-zoo Data Import . . . . . . . . . . . . . . . . . . . . . . . . 167
A.1.1 Import of metamodels . . . . . . . . . . . . . . . . . . . . 168
A.1.2 Import of ATL-transformations . . . . . . . . . . . . . . . 168
A.2 ESR Data Import . . . . . . . . . . . . . . . . . . . . . . . . . . 173

**B MatchBox Architecture** **175**
B.1 Architecture . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 175
B.2 Combination Methods . . . . . . . . . . . . . . . . . . . . . . . 175

**Bibliography** **179**

# Chapter 1

# Introduction

Data integration has been, and still is, a challenge for applications processing multiple heterogeneous data sources. Across the domains of schemas, ontologies, and metamodels this heterogeneity inevitably imposes the need for mapping specifications. Thereby, a mapping specification requires the task of creating semantic correspondences between elements to integrate multiple data sources.

An industrial example for the integration of multiple data sources is given through the point of sale scenario [134], where data coming from several retail stores needs to be integrated in a central system. A retail store has cashiers processing sales using tills and a local system collecting all data. The data from the sales of products is sent to and aggregated in a central Enterprise Resource Planning (ERP) system [136]. Thereby, the central system and third-party systems of different stores naturally differ in their internal formats which may be defined using schemas, ontologies, or metamodels. To integrate the systems a mapping between the concepts of the third-party systems and the central ERP is needed. For instance, two different representations for a purchase order or customer data have to be mapped onto each other. Support for the development of such mappings has been researched, producing matching systems that automatically propose mapping suggestions, e. g. in schema matching [126], ontology matching [33] and metamodel matching [98].

In the three matching domains most of the proposed systems claim an overall result of automatically finding nearly complete mappings, e. g. in [94, 25, 37, 38, 32]. These results are, however, only possible when running on a limited data set. When applied on heterogeneous real-world data our evaluation demonstrates that established state of the art matching techniques show only approximately half of all possible mappings found. This observation has also been confirmed by the results of the ontology alignment real-world task [32, 125]. Consequently, in this thesis we identified the challenge of (1) *improving the quality of matching results*.

The challenge of improving matching quality is not restricted to metamodel matching but also applies to schema and ontology matching. The three domains differ in the way data structures are defined. Schemas allow for a tree-based definition of elements and types. Ontologies and metamodels follow a similar way with a graph-based structure, typed relations, and the notion of inheritance. The difference in structure also concerns matching which is either tree or graph-based utilizing the structures available.

In this thesis we concentrate on metamodel matching where the data is defined through metamodels using object-oriented concepts such as classes, attributes, references, etc. We focus on the area of metamodel matching and thus Model Driven Architecture (MDA) [118], since MDA constitutes an industrial standard and specifies graph-based data models. This approach is justified by two reasons, first the field of metamodel matching is relatively unexplored and provides matching on typed graphs. Second, metamodels are increasingly applied in industry, for instance in SAP [4] throughout several products [135, 9, 143] leading to the necessity for improved matching systems. However, the concepts developed in this thesis are not restricted to metamodels but can also be applied to schema and ontology matching. The general applicability of our approach is demonstrated in our evaluation by applying our concepts to metamodels and schemas.

Besides the necessity of an improvement in matching quality another challenge is raised by an increase in size and complexity of schemas and metamodels in an enterprise context. As a consequence, the task of matching these large-scale schemas and metamodels for data integration purposes has surpassed the capabilities of most matching systems [125]. The traditional approach of pair-wise element comparison leads to quadratic complexity and thus runtime and memory issues. This results in the second challenge of (2) *missing support for large-scale matching*.

The goal of this thesis is to improve the result quality of metamodel matching in terms of correctness and completeness in the context of large-scale matching tasks. The main hypothesis of this thesis is that metamodel matching and matching in general can be improved by utilizing the graph structure of metamodels. This hypothesis is confirmed and validated using real-world data, both from the domains of metamodels and business schemas.

In the following, we will discuss the problems of matching quality and scalability. Based on these problems we will state our research questions and corresponding hypotheses to conclude with our contributions. This introduction is completed by outlining our thesis chapters.

## 1.1 Quality Problem in Matching

Metamodel matching aims at the calculation of mappings between two metamodels. These mappings should be presented to a user to support him in the task of mapping specification. Those mappings are finally used to integrate heterogeneous metamodels. Consequently, metamodel matching should reduce the effort as well as errors in the process of mapping specification. Since the mappings calculated are intended to be used by a domain expert often in an enterprise context, it is essential that those mappings are, as far as possible, correct and complete. The correctness and completeness defines the quality of metamodel matching. If the mappings calculated show a low correctness and completeness, metamodel matching may even impose an additional burden to a user applying it. Therefore, quality is one of the main concerns of metamodel matching.

One reason for, on average, only about half of all mappings being found and being correct, is the insufficient amount of information contained in a metamodel, i. e. its expressiveness. A metamodel defines object-oriented structures and thus explicit information like packages, classes, relations, attributes, etc. But, a metamodel does not contain implicit information such as the meaning of terms, codes, preconditions, etc. This additional knowledge is typically only known to domain experts.

An improvement of quality can be achieved by taking additional information into account. This can be generic information available, domain-specific knowledge, configurations or pre-processing and post-processing. For instance, the work of Garces [45] proposes a domain-specific language to incorporate external knowledge in matching processes. Some systems, e. g. [28], try to capture this additional knowledge in external representations to improve matching quality. The additional knowledge is external, because metamodels do not require a user to specify that information, thus it is not part of the metamodel. Another approach is to reuse existing mappings for instance by using transitivity for mapping calculation [23] or by coverage analysis [132]. However, each of these approaches requires external or domain-specific knowledge which one cannot be taken for granted.

Another reason for deficits in matching quality is the existence of redundant information. This is due to the fact that one element may be mapped onto multiple occurrences of another, thus producing multiple mappings instead of one. This redundant information produces misleading mappings, which may present more incorrect mappings to a user.

To summarize we identified the problem of: *(1) insufficient correctness and completeness of matching results*.

## 1.2 Scalability Problem in Matching

The problem of scalability naturally arises in the context of large-scale data as confirmed by Rahm [125]. Especially in industry metamodels easily exceed the size of 1,000 elements and may contain even more than 8,000 elements. For instance, the aforementioned point of sale scenario involves metamodels of such a size, because data about stores, customers, transactions, billing, etc. has to be stored [134]. Matching two metamodels with 8,000 elements each results in 64,000,000 comparisons to be made and stored in the memory for a combination. Even with state of the art approaches this leads to issues of high memory consumption and a runtime overhead.

The memory and runtime problems result in an inacceptable system applicability and usability. High memory consumption potentially leads to an unresponsive system and in the worst case to an abortion or crash of the matching system. The runtime problem becomes especially important in case of an interactive scenario. If a user wants to obtain metamodel matching results for a given matching task of two metamodels he typically wants to receive system feedback in seconds or better milliseconds. With large-scale metamodels the system response time increases quadratically to minutes or even hours.

Only some systems tackle this problem by limiting the matching context with light-weight matching techniques [10]. However, these techniques reduce the result quality significantly and do not tackle the memory problem for metamodels of arbitrary size. Another approach in the area of schema matching is to reduce the number of comparisons but not the schema size, e. g. in COMA++ [25] and aFlood [54]. These approaches apply a partitioning on the input, splitting it into smaller subparts and based on these parts define the comparisons to be made. However, the parts are not matched independently, thus the runtime is decreased but not the memory consumption. Hence, the approaches do not solve the problems of memory consumption and unmatchable metamodels. In addition, the approaches [25, 54] are limited to schemas or ontologies, hence they are not easily applicable to metamodels. We derive the second cause problem of our thesis: a *(2) missing support for matching metamodels of large-scale size*.

## 1.3 Research Questions and Contributions

Our main objectives are to improve quality and increase scalability for metamodel matching. A solution using information additional to metamodels is not a viable choice because this information is generally not available or requires an additional effort by a domain expert to produce. Therefore, we focus on information inherent to metamodels, namely the structural graph information of a metamodel.

Structural graph information has been utilized for fix-point based similarity propagation in schema matching [107] but not for a comparison and similarity calculation solely based on structure. The reason lies in the significant computational complexity of the graph isomorphism problem. The calculation of an edge-preserving mapping between two graphs is known to belong to NP [47]. However, the complexity of (sub-)graph isomorphism calculation can be reduced by restricting a general graph structure to specific graph classes that obey certain graph properties; an observation that will prove useful when investigating our research question:

**Research question 1.** *”How can metamodel graph structure be used to improve the correctness, completeness, runtime, and memory consumption of metamodel matching?”*

The identification of general (sub-) graph isomorphism belongs to NP, therefore the complexity must be reduced. It is known that a restriction of the input graph to special classes of graphs reduces the complexity of the isomorphism calculation. The problem is to identify the type of classes which retain as much information as possible, thus the class which is the closest to the general graph.

A well-known class restricting the input are trees. Trees are used for path computations, indices, matching [162] etc. However, a tree represents only a smaller part of the original complete graph. Especially in the context of matching this imposes a drawback in terms of quality, because less context information is available for matching.

Another more general class of graphs, known since 1930, are planar graphs. Planar graphs are graphs which can be drawn in a plane without intersecting edges. They have been formally defined by Kuratowski’s Theorem [90]. These graphs allow for a reduced polynomial complexity of several general graph problems and hence address the NP-problem. Recently they have gained interest (among others) in fingerprint classification by the work of Neuhaus [113] and general graph theory by Aleksandrov [3]. Due to their results we opt for planarity as a promising property for metamodel matching.

Of course not every metamodel is a planar graph per se, but each metamodel can be transformed into a planar graph in logarithmic time [11] by removing a minimal number of edges. In case of removed edges the original and planar isomorphism problem are not equivalent, because the removed edges are not considered for the isomorphism calculation. For the special class of trees the same problem inequivalence applies.

Representing metamodels as planar graphs offer advantages over trees. The number of edges removed when transforming a metamodel into a planar graph is considerably lower for the planar graph compared to a transformation into a tree[1]. We formulate our first hypothesis in trying to answer our research question:

**Hypothesis. H 1.** *Subgraph isomorphism calculation based on planar metamodel graphs improves correctness and completeness of metamodel matching.*

The second reason for a deficit in result quality of previous approaches is, as mentioned, redundant information which produces misleading mappings. To tackle this problem we aim to identify and process such redundant information to increase the overall matching result quality. Graph theory provides established algorithms used to discover reoccurring patterns, so-called graph-mining algorithms [14]. Applying and adapting those techniques to identify the redundant information is in our opinion an area worth exploring, which leads to our next hypothesis:

**Hypothesis. H 2.** *Mining for reoccurring patterns on metamodel graphs improves correctness and completeness of metamodel matching.*

Furthermore, the scalability problem identified by us has to be tackled in order to improve memory consumption as well as to reduce runtime. To improve memory consumption and reduce runtime the matching problem has to reduce the number of comparisons and contexts, that is the size of metamodels to be matched. A common approach in graph theory is partitioning, that is the separation of a graph into independent (unconnected) subgraphs of similar size [117]. Since metamodels are graphs and we investigate planarity, we formulate the following hypothesis:

**Hypothesis. H 3.** *Partitioning of planar metamodel graphs and partition-based matching improves support for and enables arbitrary large-scale metamodel matching.*

Thereby ”support” refers to a reduction in memory consumption and runtime on a local machine. The improved support also includes matching of metamodels of arbitrary size, because partitioning enables an independent matching of maximal sized partitions in a distributed environment. The three hypotheses presented will be validated throughout our thesis. This validation results in the following four contributions for structural graph-based metamodel matching:

- C1 We propose a planar graph edit distance algorithm for improvement of matching quality by efficiently calculating graph structure similarity for metamodels.
- C2 We suggest two matching algorithms based on graph-pattern mining for detecting (1) design patterns and (2) redundantly modelled information for metamodel similarity calculation.

[1]Our results show an average of 1% removed for planarity in contrast to 19% for trees, see Sect. 7.3.

C3 To tackle the large-scale matching problem, we propose to use planar graph-based partitioning for metamodel matching. Our approach divides the input graph into subgraphs, i. e. partitions. The calculation of partition pairs is then investigated using four partition assignment algorithms. The partitioning and partition assignment reduce memory consumption and runtime of a matching system.

C4 We perform a comprehensive real-world evaluation incorporating 31 large-scale mappings from SAP business integration scenarios as well as 20 mappings from the MDA community. These data sets form our gold-standards, i. e. they are compared with the mappings obtained by our matching algorithms.

The results of our evaluation validate our claims. That means, we demonstrate the effectiveness, i. e. improvements in correctness and completeness, and efficiency, i. e. decreases in runtime and memory consumption, of our solution for graph-based metamodel matching.

## 1.4 Thesis Outline

We structure our thesis as follows: Chap. 2 describes the foundations of graph-based metamodel matching introducing metamodel matching techniques, the basics of graph theory, graph matching, graph mining, and graph partitioning. Additionally, it also defines the graph properties reducibility and planarity. Our problem analysis is given in Chap. 3 performing a root-cause analysis for large-scale metamodel matching. The problem analysis concludes with our requirements and derives our research question. Related approaches on the identified problems of matching quality and scalability are presented in Chap. 4. Thereby, we provide an overview on state of the art of matching techniques as well as strategies for large-scale matching.

Chapter 5 presents our approach on improving the matching quality by graph-based matching utilizing planarity and redundant information. Chapter 6 presents our algorithm for graph-based partitioning that tackles the scalability problem in matching.

In Chap. 7 we validate our results with our graph-based matching framework MatchBox and real-world data. The data of our comprehensive evaluation stems from the MDA community as well as from business message mappings within SAP. Using this data we validate our algorithms w.r.t. the quality and scalability improvements. Based on the results obtained we discuss the applicability and limitations of our algorithms. Finally, we summarize and conclude this thesis in Chap. 8 giving recommendations for matching oriented data model development and pointing out directions for further research.

# Chapter 2

# Background

Since our work addresses metamodel matching employing structural graph-based approaches, this chapter will introduce the fundamental areas of metamodel matching and graph theory. We give a definition of metamodel, matching, and basic graph theory concepts. The foundations of structural matching are presented by an overview on state of the art in graph matching and graph mining. We also discuss the state of the art in graph partitioning for the purpose of large-scale matching.

## 2.1 Metamodel Matching

Metamodel matching is the discovery of semantic correspondences between metamodels, i. e. the matching of metamodel elements. In the following subsections we will define both terms, metamodel and matching.

### 2.1.1 Metamodel

A metamodel is ”the shared structure, syntax, and semantics of technology and tool frameworks”. This definition is given by the OMG in the *Meta Object Facility (MOF)* [120] specification. An interpretation is that a metamodel is a prescriptive specification of a domain with the main goal of the specification of a language for metadata. This language allows to efficiently develop domain-specific solutions based on the domain specification. This view is shared by several authors such as [6, 51, 97]. Consequently, a metamodel consists of (1) abstract syntax and (2) static semantics. The (1) abstract syntax specifies the modelling elements available. The (2) static semantics define well-formedness constraints, thus defining which model elements are allowed to be composed.

Model elements are used to specify a metamodel, which itself describes a set of valid instances, the models. These relations are called the three layered architecture of metamodels [118] and are depicted in Fig. 2.1. The

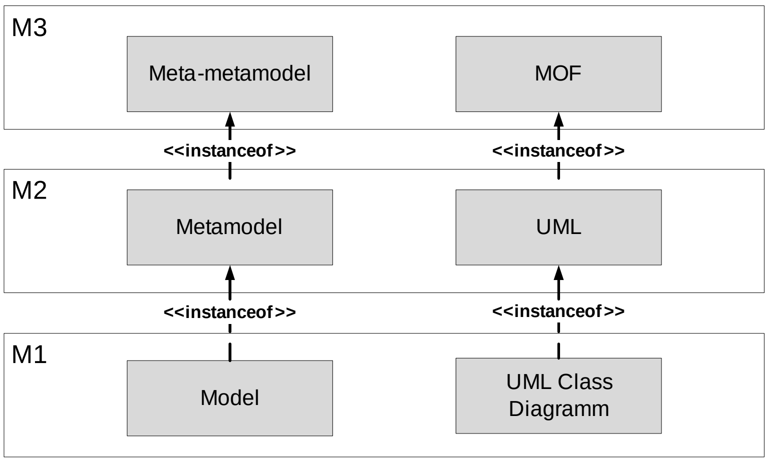


Figure 2.1: MOF three layer architecture and example

*instanceof* relation connects the different layers M1–M3, thus each element of a lower layer is an instance of an element of the upper one. A meta-metamodel on M3 defines metamodels on M2, where each of these metamodels define models on M1. On the right hand side of Figure 2.1 an example is given. MOF defines the elements available to define UML, where on M2 the UML metamodel defines which elements are available for class diagrams. Finally, on M1 a concrete class diagram can be modelled.

The constructs which MOF provides for the definition of metamodels are object-oriented constructs. A metamodel can be defined using the two main elements: packages and classes.

A package is the main container for classes, separating metamodels into modules. A class represents a type and can be instantiated. It can contain any number of attributes, references, and operations. An attribute itself has a type acting as a means for specifying values of a class' instance. Relationships between classes are represented by references and associations. An association is a binary relation between two classes, whereas a reference acts as a pointer on the associations. MOF also supports the notion of inheritance as a relation between two classes. MOF provides a range of primitive types, e. g. string or integer and it also provides the possibility of defining custom data types. A special data type is the enumeration, which allows to specify a range of values of an attribute.

The classes and other object oriented elements are used to define a metamodel, for instance UML [119], BPMN [115] or SysML [116]. A Java-based implementation of MOF is the *Eclipse Modeling Framework (EMF)* [142]. It provides the same concepts for modelling as MOF but extends them by a Java specific type system. We use EMF as the implementation and language for expressing and matching metamodels.

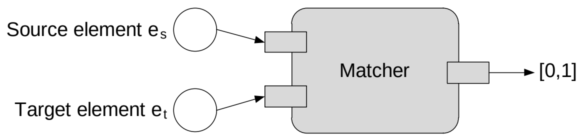


Figure 2.2: Generic matcher receiving two elements as input and calculating a corresponding similarity value as output

The definition of a metamodel by the OMG or EMF is not formal. MOF is defined verbally, where EMF is defined by its implementation. A precise and formal definition of a metamodel can be given by adopting a schema matching algebra [161]. This separation complements the classification of state of the art matching techniques. The schema matching algebra defines a schema in a generic way, thus being technical space independent[1]. We adopt this definition of schema as follows:

**Definition 1.** ***(Metamodel)*** *A metamodel $M$ is described by a signature $S = (E, R, L, F)$ where*

- $E = \{e_1, e_2, \dots, e_n\}$ *is a finite set of elements*
- $R = \{r_1, r_2, \dots, r_n\} | r \subseteq E \times E \cdots \times E$ *is the finite set of relations between elements.*
- $L = \{l_1, l_2, \dots, l_n\}$ *is a finite, constant set of labels*
- $F = \{f_1, f_2, \dots, f_n\} | f : E \times E \cdots \times E \to L$ *is the finite set of functions mapping from elements to labels*

According to this definition a metamodel comprises of elements. In case of MOF these elements are class, package, reference, attribute, operation and enumeration. The relations in case of MOF are containments, inheritance, and associations in general. The names or values of the elements and relations are labels. The definition also defines a schema with the elements: element, attribute, and type. The relations in case of schemas are limited to containment and types.

## 2.1.2 Matching

Matching is the discovery of semantic correspondences between metamodel elements, that is individuals and relations. The match operator is defined as operating on two metamodels; its output is a mapping between elements of these metamodels. Following the definition by Rahm and Bernstein [7] we define the match operator as follows:

[1]For a definition of technical space see [91].

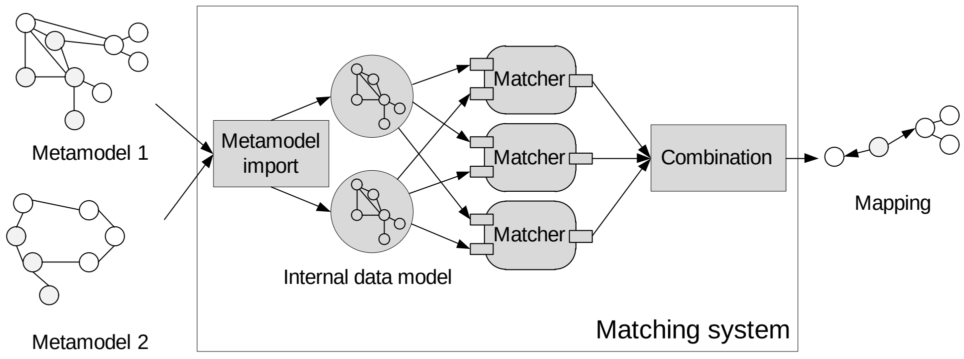


Figure 2.3: Architecture and process of a generic matching system

**Definition 2.** ***(Match)*** *The match operator is a function operating on two input metamodels $M_1$ and $M_2$. The function's output is a set of mappings between the elements of $M_1$ and $M_2$. Each mapping specifies that a set of elements of $M_1$ corresponds (matches) to a set of elements in $M_2$. The semantics of a correspondence can be described by an expression attached to the mapping.*

The match operator is realized by a matching system as described in the following.

### 2.1.2.1 Architecture of a matching system

A generic representation of a parallel matching system (e. g. [23, 24, 151]) and its components is depicted in Figure 2.3. On the left hand side two input metamodels are given, then they are processed by the matching system (match operator) and an output mapping is created. The matching system is separated into components as follows:

1. Metamodel import – transforms a metamodel into the matching system's internal data model
2. Matcher – calculates a similarity value between all pairs of elements
3. Combination – combines the matcher results to create an output mapping

**1. Metamodel import** The import component transforms a given metamodel into a matching system's internal model. Thereby, some systems apply pre-processing steps, e. g. [64, 95]. That means they exploit properties of the input metamodels to adjust the subsequent matching process. For instance, the weights of name-based techniques are adjusted if major differences in the element names of the two metamodels are detected.

**2. Matcher** A matcher calculates a semantic correspondence (match) between two elements. Unfortunately, it has been noted in several publications e.g. [126, 25], that there is no precise mathematical way of denoting a correct match between two elements. This is due to the fact that metamodels (as well as schemas and ontologies) contain insufficient information to precisely define the semantics of elements. Therefore, implementations of the match operator have to rely on heuristics approximating the notion of a correct mapping. A match is realized by a matching technique, which incorperates information such as labels, structure, types, external resources etc. We define a matching technique as follows:

**Definition 3.** ***(Matching Technique)*** *A matching technique is a function mapping input metamodel elements on a value between 0 and 1;* $f_m : E \times E \rightarrow \mathbb{R}^N$ *with* $e_s \times e_t \mapsto [0, 1]$. *This value represents the confidence defined by the function.*

An implementation of a matching technique is a matcher and therefore defined as:

**Definition 4.** ***(Matcher)*** *A matcher is an implementation of a matching technique.*

Figure 2.2 presents an abstract representation of a matcher. It depicts two given input metamodel elements (along with their corresponding context) which are processed by a matcher. Thereby, a matcher makes use of a particular matching technique to derive a similarity. In the subsequent Section 2.1.2.2 we present a classification and details on matching techniques.

**3. Combination** The combination component aggregates the results of all matchers and finally selects the output matches as mappings. Thereby, the most common way is to employ different strategies to achieve the aggregation of the matcher results [8, 25, 124, 151]. Common strategies are to average the separate results or to follow a weighted approach. Further examples are the minimum, maximum or similarity flooding [107] strategies. The aggregation can be followed by a selection which, for instance, applies a threshold for the similarity value of matches to be considered for the output mapping.

**Types of matching systems** The matching system depicted in Figure 2.3 implies a parallel execution of matchers which is not obligatory. Indeed, there are three types of matching systems, namely:

- Parallel matching systems,
- Sequential matching systems,

- Hybrid matching systems.

Parallel matching systems, e. g. [25], apply each matcher independently on the input metamodels. The matchers are executed in parallel and their result is aggregated. This approach is also followed by MatchBox [151] our proposed system for metamodel matching. In contrast, a sequential matching system, e. g. [37, 38], applies matchers one after another, i. e. a matcher's result serves as input for the following. This allows for an incremental refinement of matching results but may worsen an existing error. Finally, hybrid systems are also possible, for instance [64] use fix-point calculations by incrementally executing parallel matchers to use their results as input, again using the same matchers.

Hybrid matching systems have been generalized in meta-matching systems [123]. These systems are actually composition systems for matchers. They allow a user to specify the matcher interaction and combination to be applied. Matchers are combined via operators that have an order. This allows, for instance, for an intersection or union of matcher results, thus of matching techniques. In the following section we will classify and detail these matching techniques.

#### 2.1.2.2 Matching techniques

Several matching techniques have been proposed during the last decades originating from the areas of database schema matching, ontology matching, and metamodel matching. For a common understanding of these matching techniques and the self-containment of this thesis we provide an overview of them. The most popular classification of matching techniques has been proposed by Rahm and Bernstein in 2001 [126]. It has been refined and adopted by Shvaiko in 2007 [33] presenting a more complete and up-to-date view on matching techniques. We decided to adopt the classifications of Rahm and Shvaiko in one as outlined in [147]. The combined classification has been developed with respect to the information used for matching, e. g. names (labels) or relations.

Our classification of matching techniques is given in Figure 2.4. We call the classification adopted because we removed the class of matching techniques relying on upper level formal ontologies since it is actually a special form of reuse. We also removed the class of language-based techniques because it actually defines a specialisation of the existing class of string-based techniques. Furthermore, we refined the class of graph-based techniques thus extending the classification.

As can be seen, there are two types of classes: element level and structure-level matching. The types differentiate between techniques operating on elements and their properties and techniques using relations between the elements and thus the structure. Both classes are described in detail in the

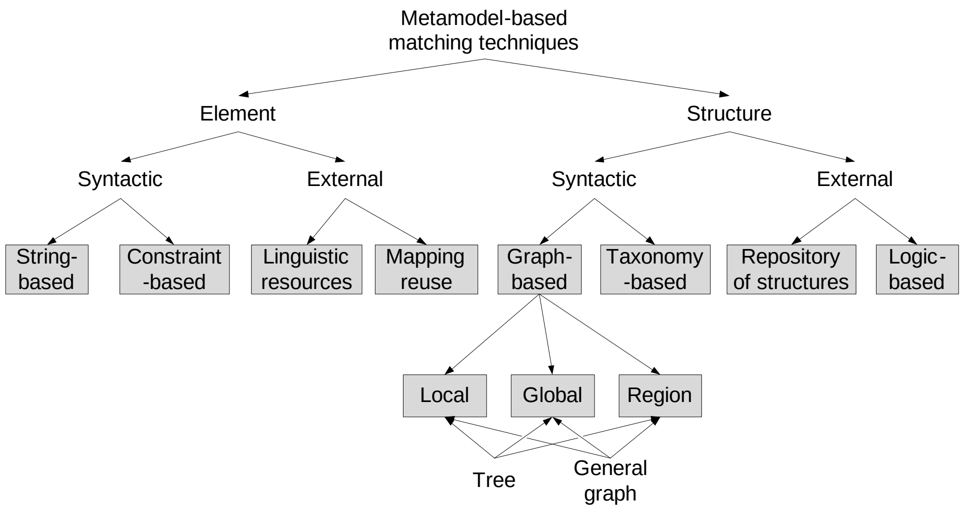


Figure 2.4: Classification of matching techniques

two following sections. Thereby, every technique class will be refined and examples for corresponding matching systems are given.

### 2.1.2.3 Element-level techniques

Element-level matching techniques make use of information available as properties of elements. In the context of metamodels and our algebraic definition, an element is an individual, thus a class, an attribute, a package, an operation, or an enumeration. An element's label is used for matching. This covers labels such as names, documentation or data types.

**String-based** String-based techniques cover similarity calculation using string information. Relevant string information includes an element's name but it also includes metadata such as documentation, annotation, etc. The techniques can be divided into the following three classes:

- Prefix-based calculation uses a common prefix as a base for a heuristics to derive a similarity value.
- Suffix-based calculation is similar to prefix-based calculation but uses a suffix instead.
- Edit-distance-based calculation aims at calculating the number of edit operations necessary to transform one string into another. The more information needed, the less similar two given names are. The most popular approach is the Levenshtein-distance [153].

- N-gram calculation targets the linguistic similarity of model elements. The element labels are split into n-character sized tokens (n-grams). For each token a similarity based on n-grams is computed, which is then the total count of equal character sequences of size n and compared to the overall number of n-grams. The resulting ratio is the string similarity.

String-based matching techniques are used by several matching systems in the form of a name matcher [24, 37, 38, 98, 107, 151] or derivations thereof.

**Constraint-based** Constraint-based matching techniques use information of elements which define a certain constraint on an element. Constraints include data types, keys, or cardinalities. Constraint-based techniques follow the rational that two elements having similar constraints should be similar. Two main classes can be separated:

- Data types are used to derive a similarity of elements based on the data type's similarity. For simple types such as integer or float a static type conversion table can be used. For complex types such as structures etc. more advanced techniques have to be applied.
- Multiplicity can be used to derive similarity. For instance, similar intervals of data types indicate a certain similarity.

**Linguistic resources** Linguistic resources are used by matching techniques relying on external sources. These external sources can be dictionaries, a common knowledge thesaurus or a domain-specific dictionary. An example for a domain-specific dictionary is a code list, encoding terms in a code as used by SAP [29]. Another popular example is WordNet [39] a publicly available dictionary used for matching.

**Mapping reuse** Mapping reuse techniques take advantage of mappings already calculated. A prerequisite is a storage for mappings which contains all mappings and the corresponding metamodels in order to reuse these mappings. The most simple approach is using transitivity as an indicator for similarity, i. e. if an element $A$ maps onto an element $B$, and $B$ maps onto an element $C$, then one may conclude that $A$ maps onto $C$. Another approach is to use existing matching techniques to derive a similarity between elements to be mapped and already mapped ones, to reuse the knowledge of their mappings.

An example for mapping reuse is COMA [24], which uses fragments that are, as a matter of fact, precisely complex types to derive mappings for the elements [23] referencing those fragments.

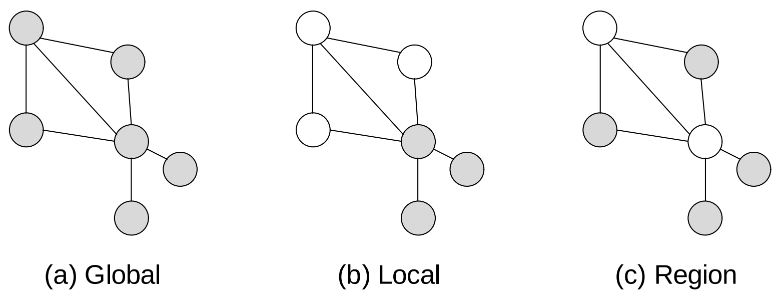


Figure 2.5: Example graph for global, local, and region-based matching context; grey highlights the elements used for matching

### 2.1.2.4 Structure-level techniques

Structure-level based matching techniques follow the rationale ”structure matters”, which is grounded in the theory of meaning [35]. Thereby, it is noted that relations between elements and their position are similar for similar elements. This structure as encoded in relations, e. g. containment or inheritance, can be used to match different elements. An important aspect of relation-ship matching techniques is the kind of graph they operate on: in the context of matching, two classes are of interest, a general graph and a tree. The following matching techniques can be applied on both. However, a general graph contains more information whereas a tree allows for optimized algorithms reducing complexity especially in terms of runtime.

We distinguish four classes of structure-level matching techniques as depicted in Fig. 2.5 [2]: global graph-based, local graph-based, region graph-based, and taxonomy-based matching.

**(a) Global graph-based** Global graph-based matching uses a complete graph in contrast to local graph-based matching, which only investigates relative elements, e. g. parent elements. Global graph-based matching techniques are either exact or inexact.

Exact algorithms describe a mapping from a vertex (element) onto another vertex as well as a mapping for edges. Subgraph isomorphism algorithms are exact algorithms. In contrast, inexact algorithms allow for an error-tolerant approach since vertices can be removed or relabelled.

- Exact algorithms, e. g. subgraph isomorphism algorithms, calculate a mapping between two metamodel graphs. The result of an exact algorithm is a mapping for each element and relation of one metamodel onto an element or relation of the other metamodel, if and only if they have the same type.

[2]A circle represents an element where an edge represents a relation, as defined in the convention of Sect. 2.2.2.

- Inexact algorithms such as the graph edit distance or maximum common subgraph algorithms apply a sequence of edit operations, composed of: add, remove, and relabel (rename). A sequence of such operations defines a mapping from one graph onto another, thus calculating the maximal common subgraph along with the operations necessary.

Global graph-based techniques have not been investigated in depth so far. However, there are selected related results, e. g. a tree-based edit distance approach by Zhang et. al [159], a simplified maximum common subgraph by Le and Kuntz [92], and an edit distance approach using expectation maximization by Doshi and Thomas [27].

**(b) Local graph-based** Local graph-based matching techniques make use of the context of an element, i. e. the relation of this element to its neighbours in a metamodel's graph. Traditional local graph-based matching techniques operate on a tree. Therefore, they use the children, leaf, sibling, and parent relationship, relative to a given element. An extension of these techniques is to generalize a graph's spanning tree and use the neighbours in the graph for matching. Examples for local graph-based techniques are the children, leaf, siblings, and parent matchers in [24, 151]. For a description of those see Sect. 7.2 in our evaluation.

**(c) Region graph-based** Region graph-based techniques make use of regions within a graph, i. e. subgraphs of the complete graph. These subgraphs are studied regarding occurences in the two metamodel graphs and regarding the subgraphs' frequency, i. e. how often they occur in the complete graph. This frequency can be used to derive a similarity between the subgraphs' elements. For instance, subgraphs sharing a high frequency are more similar. In contrast to local techniques, the context of region techniques is not restricted to a specific kind of relationship since a frequency is determined. An example of region graph-based techniques is the graph mining matcher in Section 5.2 in Chapter 5 or the filtered context matcher of COMA++ [25].

**Taxonomy-based** Taxonomy-based matching techniques operate on the special taxonomy graph in contrast to the general relationship graph. The techniques used for taxonomies are specialized in making use of the tree structure, for instance name path matching and aggregation via super or subconcept rules (parent-child relations).

**Repository of structures** The approach of a repository of structures is similar to mapping reuse. A repository contains the mappings, corresponding metamodels, and coefficients denoting similarities between the metamodels. The storage of similarities allows for a faster retrieval of mappings for a given metamodel. The coefficients are metrics such as structure name, root name, maximal path length, etc. These numbers act as an index for a set of metamodels, which allows for an efficient retrieval.

**Logic-based** Logic-based matching techniques make use of additional constraints defined on metamodels. This covers conditions defined over the metamodels as well as conditions applied to the metamodels. The matching is based on constraints in a logic language, or performed via post processing by adding reasoned mappings. For instance, consider a mapping between attributes, then a mapping between the containing classes has to exist, because attributes need a containing element. Adding this mapping is an example of logic-based matching.

## 2.2 Graph Theory

In this section we introduce basic terms such as graphs and labelled graph. The basic terms are followed by a discussion of metamodel graph representations. Subsequently, we define special graph properties which are useful for matching and partitioning and provide the foundations of the fields of graph matching, graph mining, and graph partitioning.

### 2.2.1 Definitions

Graphs are structures originating from the field of mathematics. They are a collection of vertices and edges, where the edges connect the vertices, thus establishing a pair-wise relation. The first to be known studying graph theory is Leonhard Euler in 1736 in his work on the "Seven Bridge of Königsberg" Problem [31]. This work has been refined further and is applied in many areas of today's computer science, e. g. in path finding problems, layouting, search computing, query optimization, etc. A graph is defined as follows:

**Definition 5.** ***(Graph)*** *A graph $G$ consists of two sets $V$ and $E$, $G = (V, E)$. $V$ is the set of vertices and $E \subseteq V \times V$ is the set of edges.*

A graph is called undirected, iff the edge set if symmetric, i. e. with $e_1 = (v_1, v_2)$ also $e_2 = (v_2, v_1)$ is in $E$. Otherwise, the vertex pairs defining an edge are ordered and the graph is called directed.

A graph is finite if the set of vertices is finite. A graph comprising an infinite set of vertices is infinite. Figure 2.6 (a) depicts an example for a graph, showing the vertices (circles) being connected by edges (lines). An

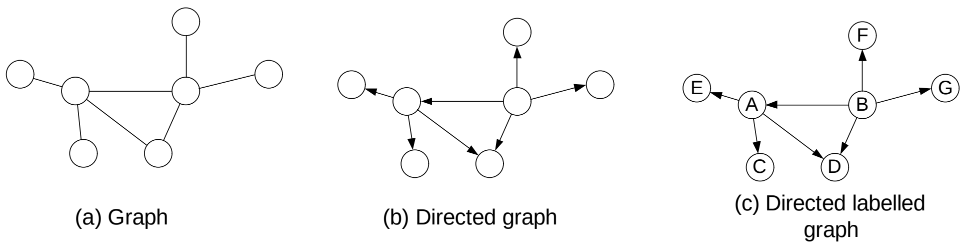


Figure 2.6: Example for a graph, a direct graph, and a directed labelled graph

example of a directed graph is given in the same Fig. 2.6 (b) adding to each edge a direction indicated by an arrow.

**Definition 6.** ***(Number of edges/vertices)*** *The number of vertices $n$ is defined as $n = |V|$. The number of edges $m$ is defined as $m = |E|$.*

In addition, each vertex and edge of a graph may have a label. A label may represent a colour, type, weight or name of a vertex or edge.

**Definition 7.** ***(Labelled Graph)*** *A labelled graph $G$, is defined as a graph and two labelling functions: $f_e : E \to L_e$ and $f_v : V \to L_v$ that map edges and vertices on edge lables and vertex labels, respectively.*

Labelled graphs are also called attributed graphs. Whenever referring to a graph in this work we refer to a labelled, undirected, finite graph. An example for a directed labelled graph is presented in Fig. 2.6 (c). Each vertex has an assigned label, in our example a name.

### 2.2.2 Metamodel representation

Ehrig et al. show in their work [30] that metamodels are equivalent to labelled, directed, finite graphs (attributed typed graphs extended by inheritance). That means for each metamodel a graph exists which has the same expressiveness and allows for the same transformations (graph operations). Even though we could treat metamodels as graphs per se, we base our observations on a metamodel's mapping on a graph to explicitly discuss the representations of relations, because we want to use the structure for matching. The first step towards a metamodel graph mapping is to separate vertex and edge mappings, where:

- A vertex mapping specifies which elements of a metamodel are represented as vertices of the metamodel's graph,
- An edge mapping defines which metamodel elements relate these vertices.

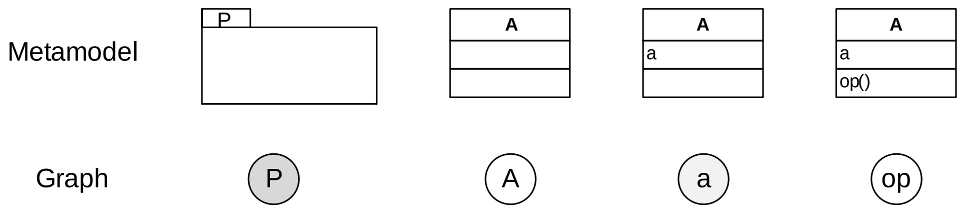


Figure 2.7: Package, class, attribute, and operation mapping onto a vertex

These mappings are based on the graphical representation of a metamodel as defined in [120] or similar in the Unified Modeling Language (UML) [121].

#### 2.2.2.1 Graph-based representation

**Vertex mapping** The elements which are mapped onto vertices are: package, class, attribute, enumeration, operation, and data type. In the metamodel's graphical representation they are represented as boxes or parts of boxes.

Figure 2.7 depicts the correspondence between the elements package, class, attribute, and operation and corresponding vertices. In the context of a labelled graph each vertex is labelled according to an element's name. An additional labelling is the type information, which can also be represented in a graph.

**Edge mapping** Edges of a graph express relations between vertices. Consequently, in a mapping between metamodels and graphs, edges may represent relationships such as inheritance, reference, and containment. These relations are also represented as edges within a metamodel's graph. The mappings of these relations can be defined as follows:

1. Inheritance can be represented explicitly by edges representing the inheritance relation or implicitly via copying all inherited members in the corresponding subclasses.

2. References and containment can be mapped onto separated edge types.

It is important to realize that the mapping between a metamodel and a graph is not unique due to different representations of inheritance relations. For example, Fig. 2.8 depicts an example metamodel and three different graph representations. The metamodel captures common scenarios, where four classes A, B, C, and D are connected by references or containments. A is related to B via bInA, B is related to D via dInB, C is contained in A via the relation cInA, and D is contained in C by dInC. Furthermore, A and D are related by an inheritance, i e. D is a subclass of A.

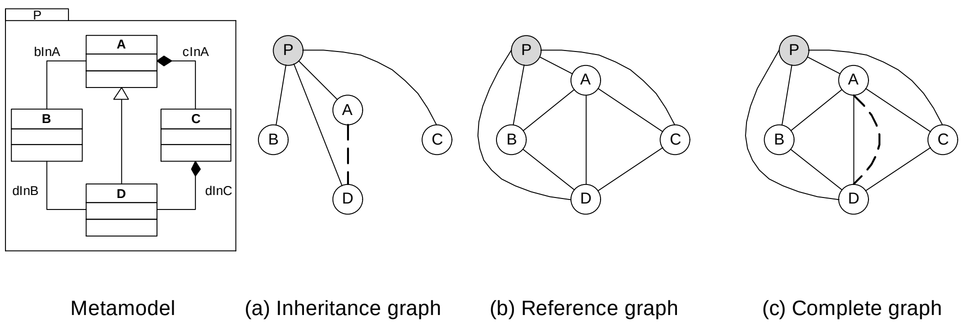


Figure 2.8: Inheritance-, reference-based, and complete graph representation of a metamodel

The first graph (a) is the inheritance-based graph, where only inheritance relations are represented as edges. According to the vertex mapping the package P and the classes A, B, C, and D are mapped onto vertices. Vertices A and B are connected by a dotted line representing the inheritance. Furthermore, all vertices are connected with the package vertex to guarantee the reachability of all vertices, thus there is an edge for each vertex to P.

The reference-based representation (b) is similar to (a), but represents the implicit containment of a package and its classes by edges. Further, A is connected to all vertices B, C, and D, and B and C are connected to D. Finally, the complete graph (c) is a combination of both representations: reference- and inheritance-based.

### 2.2.2.2 Tree-based representation

Matching techniques such as a parent, children, or leaf matcher[3] operate on a tree, because this representation allows for efficient processing of the matcher logic. A tree is a special class of a graph, where each vertex takes part in a parent-child relationship and the graph has a special root vertex. Formally, we define a tree as:

**Definition 8.** ***(Tree)*** *A graph $G$ is called a tree, if it has no simple cycle and* $|V| = |E| - 1$.

A simple cycle is a traversal of vertices where each vertex is traversed once. The number of edges relates to the number of vertices, because every vertex has at most one parent.

A metamodel does only define one trivial explicit tree. In a metamodel all elements are contained in a package and each package may contain sub-packages. This containment forms an explicit tree consisting of a package as

[3]For a detailed explanation of these matchers please see Sect. 7.2 in our evaluation.

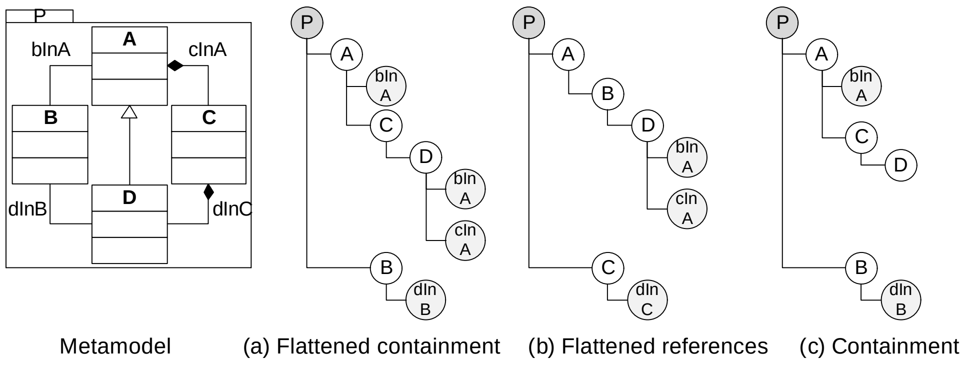


Figure 2.9: Flattened containment, flattened references, and containment tree representations of a metamodel

the root vertex and the contained elements as its children. The leaves of the tree are the attributes of the classes.

However, there are also containment relationships defined via references between classes (marked as containment relations). These relations are also part of the overall containment and should be considered in a tree as well. All of the containment references may form a graph, thus one needs to define a proper tree. In the following the tree definition of a metamodels graph is discussed. Please note that the vertex mapping is analogous to the mapping on a graph.

**Edge mapping** The problem consists of a mapping from a graph onto a tree. Several approaches to this problem can be found in literature. The closest one in mapping a meta-model onto a tree is the *Minimal Spanning Tree* (MST) of a metamodel. A MST is a subgraph of a graph connecting all vertices, thereby forming a tree with the sum of the costs of all edges being minimal. An established algorithm in order to determine the MST of a graph is Kruskal's algorithm [84].

The mapping onto a tree structure starts with a package and the elements contained. In order to compute the MST first all references and all containment references are handled as edges of a graph. An example of a metamodel mapping is depicted in Fig. 2.9, with three possible MSTs based on the three types of relations.

In Fig. 2.9 (a) the resulting tree is formed by the flattened containment hierarchy, therefore C is contained in A, and D in C, whereas B follows the containment in the package P. The flattened reference import is depicted in Fig. 2.9 (b) having the path A, B, D, because the algorithm takes the left side path first (assuming these elements have been created first). Since we are calculating the MST, no element will have multiple occurrences in the resulting tree.

Besides the problem of defining a proper tree structure, inheritance has to be dealt with. A common approach ignores the semantics of inheritance and discards this information. Alternatively, a metamodel can be *flattened*, i. e. copying all attributes and relations of a super class into its subclasses. This approach preserves the information defined by inheritance, but loses the connection between super and subclasses. In Fig. 2.9 (a) and (b) a flattened import is shown based on the reference and containment hierarchy, whereas (c) shows a non-flattened import, because D does not contain the references inherited from A.

### 2.2.3 Graph properties

In this section we introduce the graph properties reducibility and planarity. Reducibility is used by our redundancy matcher and planarity in our planar graph-based edit distance matcher as well as in the planar graph-based partitioning.

#### 2.2.3.1 Reducibility

Reducibility describes the behaviour of a graph under edge contraction. Edge contraction defines the merging of vertices and their edges. If under edge contraction the graph can be contracted into a single vertex, the graph is called reducible. Formally, edge contraction is defined as:

**Definition 9.** ***(Edge contraction)*** *Let the graph $G = (V, E)$ contain an edge $e = (u, v)$ with $u \neq v$. Let $f$ be a function which maps every vertex in $V \setminus \{u, v\}$ to itself, and $\{u, v\}$ to a new vertex $w$.*

- *The contraction of $e$ results in a new graph $G' = (V', E')$, where $V' = (V \setminus \{u, v\}) \cup w$, $E' = E \setminus \{e\}$, and*
- *For every $x \in V$, $x' = f(x) \in V'$ is incident to an edge $e' \in E'$ if and only if the corresponding edge $e \in E$ is incident to $x$ in $G$.*

The foundation of reducibility are graph minors, which are based on the principle of edge contraction. A minor of a graph is defined as follows:

**Definition 10.** ***(Graph minor)*** *An undirected graph $H$ is called a minor of the graph $G$ if $H$ is isomorphic to a graph obtained by zero or more edge contractions on a subgraph of $G$.*

A minor $H$ of $G$ thus results from any sequence of edge contraction operations on a graph $H$ that lead to a subgraph of $G$. The edge contraction is applied on a particular edge by removing it and merging it with its incident vertices. This operation can be applied on a set of edges in any order.

Figure 2.10 shows a minor of a graph along with the edge contraction necessary. The graph in (b) is a minor of the graph in (a), because a sequence

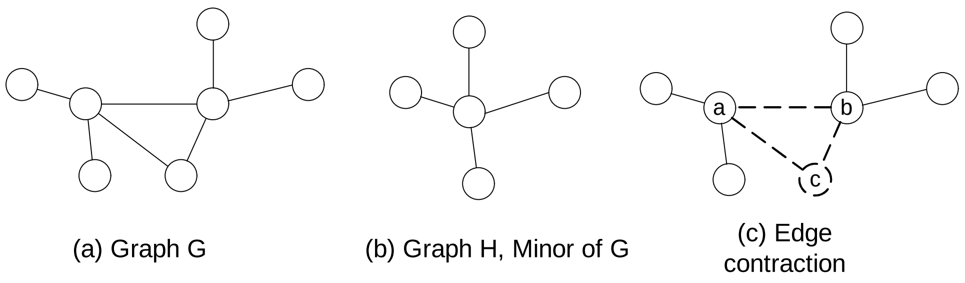


Figure 2.10: Example of a graph, a minor of this graph, and the corresponding edge contraction

of edge contraction operations can be defined to form a subgraph. These operations are shown in Fig. 2.10 (c), where three vertices are labelled to explain the operations. First, the edge connecting the vertices $a$ and $c$ is removed and both vertices are merged. The result is a multi-edge between $a$ and $b$. Second, one edge of the multi-edge is removed and both vertices are merged. The remaining multi-edge becomes a self-edge of $a$. Finally, the self-edge is deleted. The resulting graph $H$ is a minor of $G$.

This sequence of edge contraction has also been used in the area of compilers [2]. In this field the analysis of control flow graphs requests for graph (tree) operations such as edge contraction, too. Thereby, use is made of two special transformations, $T1$ and $T2$. Applying them on a graph in any order defines the reducibility of a graph:

**Definition 11.** ***Reducibility*** *Let $G = (V, E)$ be a directed graph where the following transformations are defined:*

- *$T1$: Remove a self-edge $(v, v)$ in G.*
- *$T2$: If $(v, w)$ is the only edge entering $w$ and $w \neq v$ delete $w$. Replace all edges $(w, x)$ by a new edge $(v, x)$ and additionally all edges $(x, w)$ by an edge $(x, v)$.*

*A graph is called reducible, if and only if it can be transformed to a single vertex by applying the two transformations T1 and T2. Otherwise the graph is called irreducible.*

#### 2.2.3.2 Planarity

The planarity property of a graph defines when to call a graph planar. It has not been considered for metamodel matching so far. However, planarity allows for interesting applications, since graph algorithms requiring planarity can reduce some NP-complete problems to a polynomial, mostly quadratic,

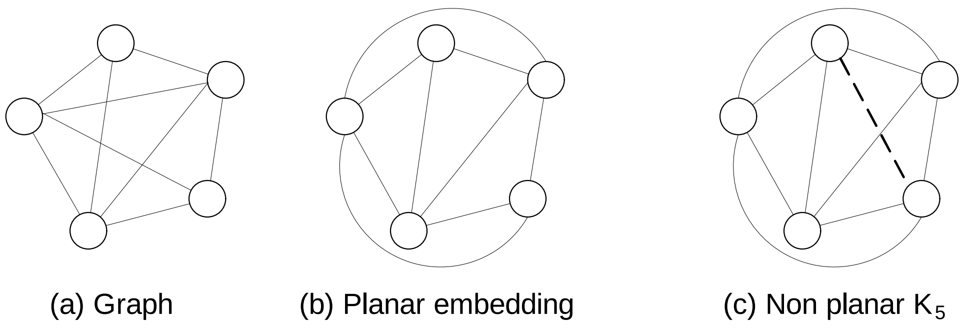


Figure 2.11: Example of a graph, its planar embedding, and non-planar extension

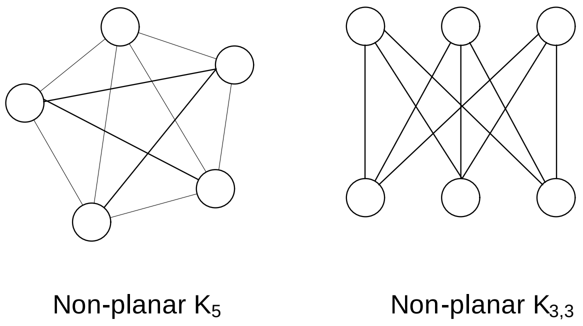


Figure 2.12: The non-planar graphs $K_5$ and $K_{3,3}$

runtime when applied on planar graphs. For instance, identifying graph similarity by subgraph isomorphism calculation on general graphs is known to be NP-complete. If the input graphs are restricted to be planar, the problem can be solved in almost quadratic time. We make use of this property for our planar graph edit distance matcher (Chap. 5) and our planar partitioning (Chap. 6).

An illustrative description of planarity is: A graph is planar, if such drawing exists, that none of the graph edges intersect. That means, a graph is planar if it can be embedded into a plane without intersecting edges. This process creates a planar embedding. Figure 2.11 depicts an example for a graph (a) and the corresponding planar embedding (b). As can be seen the example graph is planar. However, if the graph is extended as shown by the dashed line in (c), the graph becomes non-planar, because there is no drawing without intersecting edges.

The graph including the dashed line in Fig. 2.11 (c) is a special graph called $K_5$, which is one of the two basic non-planar graphs. The other basic graph is called $K_{3,3}$ and consists of six vertices, which are arranged in two lines of three vertices each with edges between all opposing vertices. Figure 2.12 depicts both graphs.

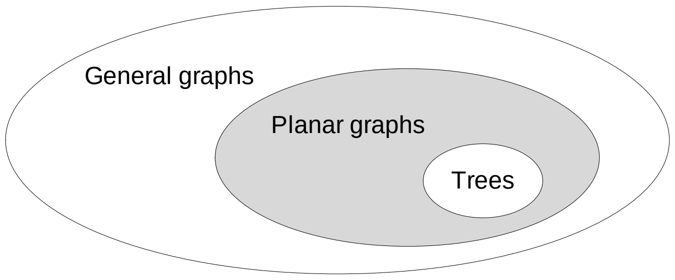


Figure 2.13: Subset relation between general graphs, planar graphs, and trees

These graphs are essential for the formal definition of planar graphs, which has been done 1930 in Kuratowski's theorem [90]. He states:

**Theorem 1.** *A finite graph is planar if and only if it does not contain a subgraph that is a subdivision of $K_5$ (the complete graph on five vertices) or $K_{3,3}$ (complete bipartite graph on six vertices, three of which connect to each of the other three).*

Thereby, a subdivision is the result of a vertex insertion in an edge. That means, an edge is split into two edges via a new vertex, still the two new edges connect the original vertices via the new vertex. Instead of using subdivisions their counterpart, minors, can be used for defining planarity. In 1937 Wagner's conjecture [152] has been presented and proved in 2004 by Robertson and Seymour [131]. It states:

**Theorem 2.** ***(Planar)** A finite graph is planar if and only if it does not include $K_5$ or $K_{3,3}$ as a minor.*

According to the definition of a minor, which is a contraction of vertices and their edges, the theorem states, that the graph must not be reducible on either one of the non-planar graphs, $K_5$ and $K_{3,3}$. Consequently, every tree is also a planar graph.

The relation between general graphs, planar graphs, and trees is depicted in Fig. 2.13. The set notation shows the subset relation between the special classes of graphs. Each tree is also a planar graph, where each planar graph (and tree) is naturally a general graph, whereas not every graph is planar or a tree.

The question arises if metamodels are planar and if not, how to make them planar. Both theorems are not suited for a planarity check implementation, but there are algorithms for doing the check with a linear complexity. The most popular one will be described followed by an algorithm to make metamodels planar.

#### 2.2.3.3 Planarity check for metamodels

Metamodels are not planar per se, therefore a planarity check is needed. The planarity check is well-known in graph theory, so we selected in this thesis an established algorithm by Hopfcroft and Tarjan [61], because it has the lowest runtime complexity ($O(n)$).

The idea of the algorithm is to divide a graph into bi-connected components (cf. [20], page 11), which are tested for their planarity. Hopfcroft and Tarjan have shown that a planarity of all components results in planarity for the complete graph. The test for each component is done by detecting cycles in them. Each cycle is arranged as a path, where non-cycle edges have to be arranged left-hand or right-hand side. Both groups have to contain non-interlacing edges. If for a given edge no group can be found without violating the non-interlacing property, the graph is non-planar. For further details refer to [61].

#### 2.2.3.4 Maximal planar subgraph for metamodels

If a planarity check fails, a metamodel needs to be made planar in order to take advantage of the planarity property. The planarity property can be established by removing vertices or edges. To solve the problem an approach exists where the number of edges is maximal. That means, if an edge that has been previously removed would be re-added, the graph would be non-planar. This algorithm has been proposed by Cai et al. [11] resulting in a complexity of $O(m \log n)$ ($m$ is the number of edges and $n$ of vertices).

The idea of the algorithm is to recursively compute planar subgraphs of all successors of a particular edge. Afterwards, the subgraphs are combined into one graph by deleting planarity-violating edges. The approach by Cai et al. tests every edge whereas Hopfcroft and Tarjan consider all paths. Furthermore, the algorithm by Cai et al.uses a so-called attachment, which is a set of blocks grouping non-interlacing non-cycle edges. The attachments are used to determine the edges to be removed by testing for planarity. Finally, the attachments are recursively merged to construct the maximal planar subgraph. For further details please refer to [11].

The algorithms allow checking any given metamodel for planarity and performing planarisation if necessary. The overall complexity of both operations is $O(m \log n)$ ($m$ is the number of edges and $n$ of vertices).

### 2.2.4 Graph matching

Graph matching is the similarity calculation of two input graphs. It has first been treated as a mathematical problem but was adopted in several application domains such as pattern recognition and computer vision, computer-aided design, image processing, graph grammars, graph transformation, and

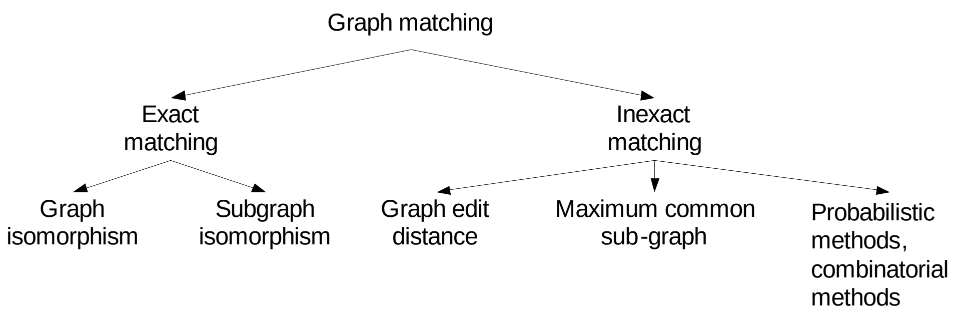


Figure 2.14: Classification of graph matching algorithms adopted from [145]

bio computing [145]. Conceptually, graph matching can be seen as the decision problem whether a graph $H$ contains a subgraph isomorphic to a graph $G$, i. e. if both graphs share similarities. This condition can be relaxed to identifying a subgraph contained in both graphs and even to graphs with a certain distance. Unfortunately, the problem of finding a graph isomorphism for general graphs belongs to NP and is either in NP-complete or P [127]. Therefore, approximations or restrictions on graph properties can be used to cope with the complexity problem. To solve the aforementioned questions so-called graph matching algorithms have been proposed.

#### 2.2.4.1 Overview of graph matching algorithms

Graph matching algorithms can be divided into two classes: exact and inexact algorithms. Exact algorithms aim at a subgraph calculation whereas inexact algorithms are error-tolerant and allow certain distances between the subgraphs. Figure 2.14 depicts a classification provided by [145]. The subsequent paragraphs deal with the exact, i. e. graph and subgraph isomorphism algorithms, and inexact algorithms, i. e. graph edit distance, maximum common subgraph and combinatorial methods, in detail.

**Exact matching** Exact matching algorithms aim at calculating for two given input graphs a one-to-one mapping between their vertices and edges. The resulting mapping is a graph isomorphism which is defined as follows:

**Definition 12.** ***((Sub-)Graph Isomorphism)****. Given a graph $G_s = (V_s, E_s)$ as source and $G_t = (V_t, E_t)$ as target a graph isomorphism is defined by a function $f : V_s \to V_t$ such that for every edge $e_s = (v, w)$ also $e_t = (f(v), f(w)) \in E_t$. If $|V_s| < |V_t|$, then f is called a subgraph isomorphism.*

The standard approach on subgraph isomorphism identification is based on backtracking and has been proposed by Ullmann [144] in 1976. Despite being widely referenced the algorithm lacks scalability. That means it is only applicable for a rather small number of vertices in the graphs compared (less than 20). Our experiments showed that even at a size of 15 elements for metamodels to be matched the runtime rises to minutes. This is due to a search space explosion for graph isomorphism calculation, because every possible edge/vertex combination needs to be investigated.

**Inexact matching** Inexact matching approaches relax the edge preservation condition of graph isomorphism identification. That means edge mappings are not necessarily required if vertex mappings have been found. One approach on inexact matching is defined by the graph edit distance proposed 1983 by Sanfelui [133]. We define the graph edit distance as follows:

**Definition 13.** ***(Graph Edit Distance)****. Let $G_s$ and $G_t$ be two graphs. The Graph Edit Distance (GED) is defined as a finite sequence of edit operations leading to an isomorphism of $G_s$ and $G_t$. The edit operations are addition, deletion, or relabelling of a vertex or edge.*

Several approaches for a calculation of the graph edit distance have been proposed, a survey of graph edit distance algorithms can be found in [44]. Still the problem of graph edit distance calculation for general graphs remains NP-complete [44], because every vertex of the source graph can be mapped on every vertex of the target graph with different edit distances. Therefore, general edit distance algorithms are not applicable for large graphs [110], but again approximate or input restricting algorithms can be applied.

### 2.2.5 Graph mining

The essential task of graph mining algorithms is to discover frequent subgraphs (patterns) in one or more graphs. Mining algorithms have been used in the domains of bioinformatics for the discovery of frequent chemical fragments and classification [103], VLSI reverse engineering [89] and in general for tasks of deriving association rules due to similar patterns [14].

These applications have in common that they can be reduced to the problem of finding reocurring subgraphs. We define a frequent subgraph as follows.

**Definition 14.** ***(Frequent Subgraph)*** *Given a subgraph $S(V', E')$ of a graph $G(V, E)$ with $V' \subseteq V$ and $E' \subseteq E$ and a function $f : G \rightarrow \mathbb{N}$. The frequency of $S$ is defined by $f(S)$. If $f(S) > t, t \in \mathbb{N}$ then $S$ is called frequent.*

A subgraph has to occur more than $t$ times in a graph to be called frequent. We define such frequent subgraphs as patterns.

**Definition 15.** ***(Pattern)*** *A pattern $P$ is a frequent subgraph.*

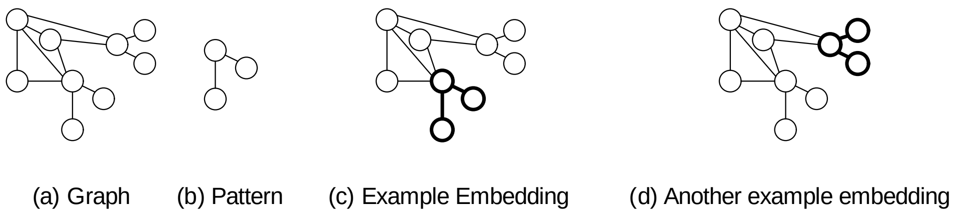


Figure 2.15: Example of a graph, a pattern and embeddings of this pattern

According to this definition, each pattern has one or more occurrences. We call these occurrences embeddings.

**Definition 16.** ***(Embedding)*** *If a subgraph $S$ of $G$ exists so that $S$ is isomorphic to a pattern $P$, then $S$ is an embedding of $P$.*

For clarification Fig. 2.15 depicts examples for the previously defined terms. On the left (a) a graph is shown with (b) a possible pattern. An embedding of this pattern is shown in (c). Our example pattern has to have more than one embedding to be frequent, accordingly we display another possible embedding in (d).

According to [89], there are two distinct settings classifying the mining algorithms w.r.t. their scenario, namely the single graph setting and the graph transaction setting:

- *Single graph setting* defines an extraction of patterns in one graph where the frequency of a pattern is the number of embeddings in this graph.
- *Graph transaction setting* defines an extraction of patterns on a number of graphs. The frequency of a pattern is thereby determined by the number of graphs which have at least one embedding of this pattern.

Please note that single graph setting algorithms are not applicable for graph transaction scenarios but graph transaction algorithms for single graph settings. We depict the two settings in our classification in Fig. 2.16 with two additional dimensions *approximate* and *complete* which has been introduced in [89]. Since the main complexity of the algorithms is due to the subgraph isomorphism tests they can be separated into complete and approximate algorithms. Complete mining algorithms are complete in the sense that they are guaranteed to discover all frequent subgraphs, that is patterns. In contrast, approximate algorithms calculate a subset of the complete set of all patterns and thus not the optimal, i. e. complete, solution.

### 2.2.6 Graph partitioning and clustering

Graph partitioning and graph clustering both deal with the problem of splitting a graph into smaller subgraphs. Graph clustering algorithms try to op-

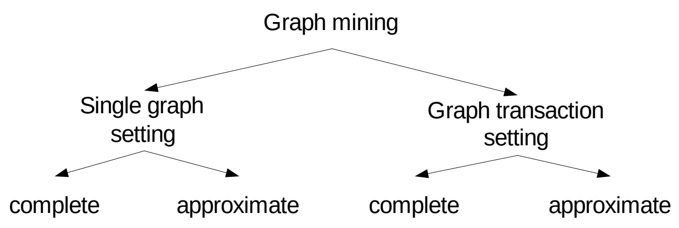


Figure 2.16: Classification of graph mining algorithms

timize the clusters calculated w.r.t. an a priori defined quality criterion. In contrast, graph partitioning aims at calculating subgraphs of nearly equal size in the context of a weighted graph. We define the graph paritioning problem as given in [117] as follows:

**Definition 17.** ***(Graph partitioning)*** *Given a weighted graph $G = (V, E)$ and a positive integer $p$, the problem of graph partitioning consists of finding $p$ subsets $V_1$, $V_2$, ... $V_p$ of $V$ with $i, j \in \{1, \dots, p\}$ such that*

1. $\bigcup_{i=1\dots p} V_i = V$ *and* $V_i \subseteq V$, $V_i \neq \emptyset$ *and* $V_i \cap V_j = \emptyset$ *for* $i \neq j$
2. $w(V_i) \approx \frac{w(V)}{p}$, *where $w(V_i)$ and $w(V)$ are the sums of the vertex weights in $V_i$ and $V$, respectively, and $\approx$ allows for small derivations in size,*
3. *The cut size, i. e. the sum of the weights of edges crossing between the subsets is minimized.*

The goal of graph partitioning is the calculation of a given number of subgraphs balanced in their weight. Each subset $V_i$ of Def. 17 and the corresponding edges are a partition.

**Definition 18.** ***(Partition)*** *Given a graph $G = (V, E)$ each subgraph $S_i \subseteq G$ is a partition.*

Thus a partition not only defines a subset of a graph but requires each vertex of the graph to be only part of one partition, hence partitions are disjoint. Partitioning algorithms try to find a minimal number of vertices or edges being removed from a graph such that the resulting vertices or edges form partitions.

We give a classification of hierarchical graph clustering and graph partitioning algorithms in Fig. 2.17. Algorithms for graph splitting are either hierarchical graph clustering or graph partitioning algorithms each separated into local and global approaches.

Local graph clustering approaches begin with assigning each vertex to a different cluster. Then, these clusters are merged until a given criterion is reached. Approaches following this behaviour are called agglomerative.

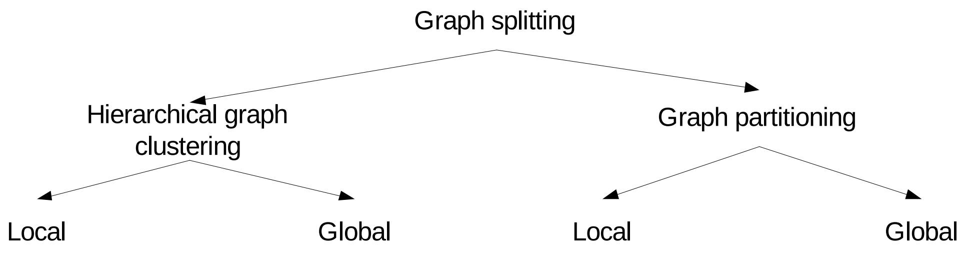


Figure 2.17: Classification of graph partitioning and clustering algorithms; adopted from [117]

Representatives are given in the modularity approach by Newman [114] or density-based clustering [93]. In contrast, there are divisive clustering approaches which begin with the complete graph, splitting it recursively until again a given criterion is fulfilled. Examples for global clustering are Betweeness [48] or clustering based on the Kirchhoff equations [155].

Local graph partitioning approaches are similar to local clustering algoriths, the most popular one is the greedy approach by Kernighan and Lin [78]. Their approach has been adopted in hMetis [75] transforming it into a gobal multilevel approach. Another example for global partitioning is bisection, which recursivly splits a graph, by using the vertex distances [111]. For a more detailed and extensive survey of graph partitioning approaches please refer to [117].

## 2.3 Summary

We have introduced the fundamental concepts of metamodel matching and graph theory. The key findings of this chapter may be summarized as follows:

**Metamodel matching** We defined a metamodel as the prescriptive specifications of a domain which specifies a language for metadata. We described the MOF-standard as a meta-metamodel defining object-oriented concepts like classes, attributes, and relations such as inheritance or references. A formal definition of a metamodel was given, defining a metamodel as a set of individuals, labels, labelling functions, and relations. We then defined the match operator, which is applicable on two metamodels creating a mapping between elements of these two metamodels. Presenting a common matching system architecture, we also defined a matching technique as a function assigning a similarity value for two given metamodel elements and we presented a classification of the state of the art of matching techniques. These classes are separated into element-level techniques, e. g. string-based, and structure-level techniques, e. g. graph matching.

**Graph theory** We gave an overview of the basics of graph theory, defining a graph as a set of vertices connected by edges. Subsequently, we defined a directed graph as a graph with directed edges and a labelled graph as a graph with a labelling function, assigning a label to vertices and edges. Finally, we stated that we use a directed, labelled, and finite graph in this thesis. We also introduced the terms and gave a short overview on the state of the art for graph matching, graph mining, and graph partitioning.

**Reducibility and planarity** We defined reducibility as a special graph property, which is used in our matching and partitioning approaches. Reducibility defines the edge contraction operation on a graph, i e. the deletion of edges and merging of corresponding vertices leading to a hierarchical graph.

The graph property planarity allows efficient algorithms to be applied in case of $NP$-complete subgraph isomorphism and partitioning problems. We gave a definition of planarity, stating that a graph is planar if it cannot be reduced to the special graphs $K_5$ and $K_{3,3}$ or more illustratively, if a graph can be drawn into a plane without any edge intersection.

# Chapter 3

# Problem Analysis

The core problem addressed in this thesis is insufficient quality of matching and insufficient support of scalability. In this chapter we present our structured problem analysis to demonstrate the problems we tackle with our work and to define the scope of our work. We begin with an illustrative example for the core problem. Then we describe our methodology for analysing the problem followed by the root-cause analysis, and the objectives which lead to the resulting requirements of our solution. The relation to the requirements is established by presenting our systematic approach for developing a solution for the problems analyzed and our research question.

## 3.1 Motivating Example

Our motivating example originates from the area of retail stores and is an official SAP scenario [134]. A retail store has cashiers processing sales using tills and a local system collecting all data. This data is sent to and aggregated in a central Enterprise Resource Planning (ERP) system. Since SAP does not produce tills and associated systems, the central system and third-party systems of different stores need to be integrated. The subsequent section deals with a refined description of the scenario motivating the problem of data integration of different formats and thus metamodels. The related metamodels are described in Sect. 3.1.2, followed by an exemplary description of problems in applying matching on a large-scale scenario in Sect. 3.1.3.

### 3.1.1 Retail scenario description

The retail scenario describes an integration scenario between a Point of Sale (POS) system and an Enterprise Resource Planning (ERP) system. A POS system is a third-party system located in a retail store having an own user interface tailored to support cashiers. A POS system is specialized to the

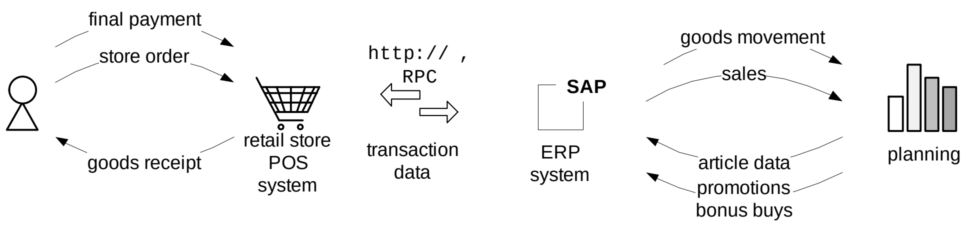


Figure 3.1: Example of data integration in case of message exchange between a retail store (POS system) and an ERP system

sales process of a customer, thus it needs to be supplied with master data such as article data as well as promotion and bonus buys.

Complementarily, an ERP system includes functionality to plan and manage the business of a company. It provides means for article data management, planning, ordering, goods movement, report generation, etc.

Figure 3.1 illustrates the retail store scenario. On the left hand side a customer interacting with a store and the corresponding POS system is shown. On the right hand side, the company is represented by its ERP system and a graph representing a report, indicating the planning facilities provided by the ERP system. In the middle data is exchanged between the ERP and the POS system. For instance, the data can be related to goods movement or sales from the POS to the ERP or article data as well as promotion and bonus buys. According to [134] the data flow contains:

- Data transfer from the ERP to the POS system
  - Article and price data: inventoried and value-only article types in various article categories (single article, generic article ...)
  - Bonus buys data (with requirements and conditions necessary to determine a specific deal)
  - Promotion data (with special price, start and end date of promotion)
- Transfer of data from the POS system to the ERP
  - Sales and returns: article purchases or returns at the POS
  - Tendering: legally valid means of payment used at the POS
  - Financial transactions: involve monetary flow at the POS without movements of goods (e.g. cash withdrawals)
  - Totals transactions: represent information on the balancing of registers and stores
  - Control transactions: technical information about behaviour of the POS (e.g. cash drawer opening / closing)

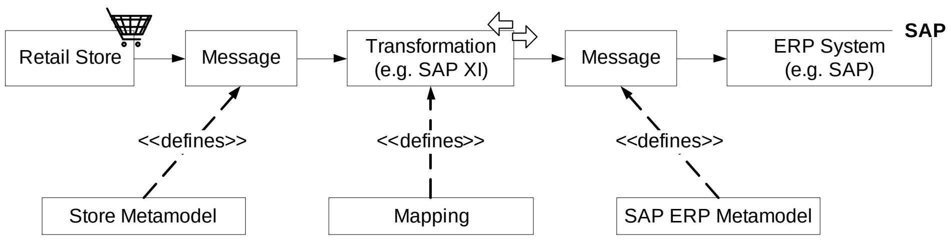


Figure 3.2: Details of retail store data integration example

The data exchange of both systems is depicted in more detail in Fig. 3.2. The third party POS and the SAP ERP system exchange messages in different formats to communicate. These formats need to be transformed into each other to enable the different systems to process them. The design time challenge is the integration of both data formats, i. e. the integration of both metamodels.

This metamodel integration is achieved by specifying a mapping between the metamodels' elements. This could be done manually, however, in our case the source metamodel constitutes 971 elements, where the target metamodel has 3,775 elements. A mapping specification will easily require weeks to be completed [143], so this task is time-consuming and error-prone as also identified in several publications, e. g. in [8, 37, 151]. The required assistance is provided by metamodel matching as will be discussed in the following section.

### 3.1.2 ERP and POS metamodels

In case of two different metamodels defining two different specifications of similar entities, a data integration problem arises. In our example these specifications are messages exchanged between the POS and ERP system with respect to sales, good movement, etc. Both systems have been developed for a special purpose and by different vendors, the third-party POS for supporting cashiers and the SAP ERP for planning the whole business, and therefore both systems have different schemas tailored to their purpose.

The metamodel of a POS system needs to capture information about transactions, article purchases, payments, cashier withdrawals, etc. An ERP system's metamodel needs to define all this information and additional data about stores, bonuses, etc. In our scenario the POS metamodel consists of 3,775 elements, i. e. classes and attributes. The metamodel of an ERP for retail contains 971 elements. Figure 3.3 depicts excerpts of these metamodels, the POS metamodel on the left and the ERP metamodel on the right.

The POS excerpt starts with the *RetailTransaction* containing three elements: *RetailLoyalty*, *RetailCustomer*, and *TransactionItem*. The RetailLoyalty describes the bonus programme involvement of a customer, i. e. the

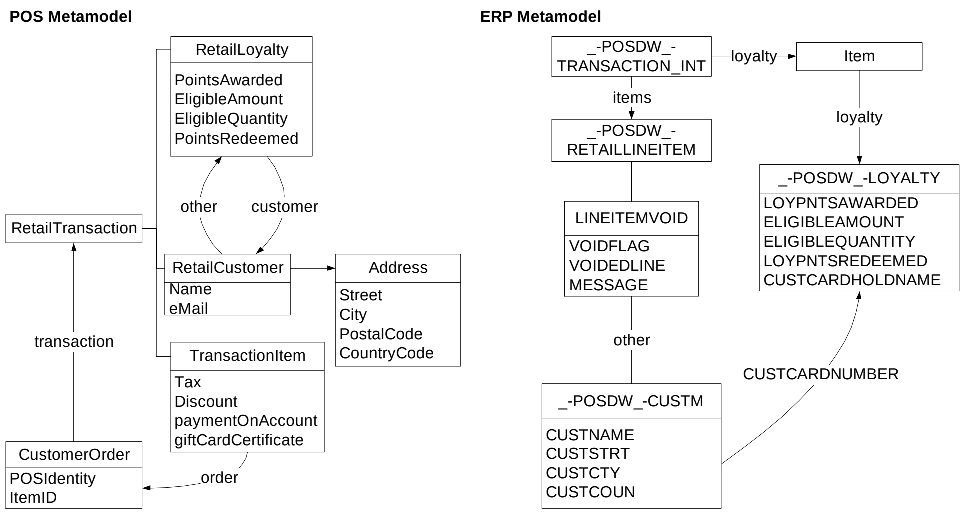


Figure 3.3: Example of a transaction in a retail store metamodel and a central ERP system metamodel

points awarded, the eligible amount and quantity of points, and the redeemed points. Thereby, each RetailLoyalty element is assigned to a customer and vice versa. A *RetailCustomer* defines a customer by his name and email. Furthermore, each customer has an address consisting of a street, city, postal code, and country code. The last element of a RetailTransaction is the transaction itself, i. e. the *TransactionItem* element, which contains information about the tax, possible granted discount, payment information, and optional gift card certificates. This transaction is assigned to an order. This *CustomerOrder* has two identifiers and relates to the transaction element.

The ERP excerpt follows a different naming used in SAP ERP systems originating from a restricted length of identifiers in old ERP systems. A company internal transaction is described by the *_-POSDW_-TRANSACTION_INT* element. It relates to two elements, the retail line item as information about the purchase and the item element acting as a proxy for the *_-POSDW_-LOYALTY* element. The loyalty captures information about the points awarded, the eligible amount and quantity, redeemed points, and additionally the name of the customer card holder's name. Each transaction also consists of a *_-POSDW_-RETAILLINEITEM* which has *LINEITEMVOID* acting as proxy for the associated customer. A *_-POSDW_-CUSTM* has a name, street, city, and country.

### 3.1.3 Data integration problems

The data integration problem of both metamodels is the problem of defining a mapping between the POS and the ERP metamodels. The mapping defines the correspondences between the metamodels which in our example is straight forward. A customer, transaction item, and loyalty each have corresponding elements. However, considering the real world scale of this problem, i. e. more than 3,000 elements, this task requires a lot of time for specifying and checking the mappings.

Metamodel matching assists in this task by calculating the element correspondences. The most common approach is to use the name similarity of elements. Considering our example, this works for the loyalty items, because the names *RetailLoyalty* and *_-POSWD_-LOYALTY* are similar to each other. This is done by using a string edit distance, e. g. refer to [153]. However, the customer elements will be hard to be mapped using name-based similarity. A type-based approach may assist by discovering the mapping between both country code elements (*countryCode* and *CUSTCOUN*). Still, the proxy objects such as *Item* or *LINEITEMVOID* will not be discovered, also references such as *CUSTCARDNUMBER* are hard to find. Consequently, the mappings discovered by matching tend to be incomplete.

Another problem is the discovery of incorrect mappings. For instance, consider the relation named *other* between *RetailCustomer* and *RetailLoyalty* in the POS metamodel. There exists an equally named relation in the ERP metamodel, but it relates *_-POSDW_-CUSTM* and *LINEITEMVOID*. Therefore, the mapping discovered solely on names is incorrect. A possible solution is a combination of type and name information as well as the consideration of structural properties based on trees.

Consequently, metamodel matching may produce incomplete and partially incorrect results. That leads to the conclusion that the matching task lacks quality and mapping calculation is a challenging task. Even with the use of all matching techniques presented in Sect. 2.1.2.2 not all mappings are found. So, the matching quality leaves room for improvement.

Another issue arises when considering the size of the matching task. A common metamodel matching system compares all pairs of source and target elements to derive the mapping. In our case these are $971 \times 3,775$ elements, which results in 3,665,525 comparisons. These comparisons have to be done for each matching technique applied, which increases the total even more. This high number of comparisons leads to a high runtime (in the range of minutes or even hours) for the matching as well as high memory consumption (in the range of gigabytes) for the given task. Considering even bigger examples of metamodels with more than 10,000 elements the matching task may be impossible due to lack of memory.

To conclude the real-world large-scale retail example shows that both matching quality and scalability can be improved. An analysis of these problems, objectives, corresponding requirements and our approach is presented in the subsequent section.

## 3.2 Problem Analysis

The problem analysis we present will justify and detail our research question on improving metamodel matching in quality and scalability by utilizing structural information. The problem analysis was carried out by applying the method of *Zielorientierte Projektplanung (ZOPP)*, i. e. Objectives-oriented Project Planning, which has been proposed by the German Technical Cooperation (GTZ) [60].

The idea of ZOPP is to begin with a problem hierarchy, which is the basis for narrowing the scope. The resulting scoped problem hierarchy is transformed to a goal hierarchy to finally define the objectives resulting from the problems. Thereby, the problem hierarchy consists of problems and subproblems, where each subproblem is a cause for a problem. In the following, we use the first section to present our problem hierarchy and define the scope. In the second section we derive our goal hierarchy and define the corresponding objectives. The third section formulates the objectives into requirements for our approach in the fourth section.

### 3.2.1 Problems and scope

The problems we identified are captured in the problem hierarchy. The problem hierarchy is a tree, where the parent child relation is defined as ”leads to” and the child parent as ”is caused by”. That means each problem is a subproblem of another one, thus forming a problem hierarchy.

#### 3.2.1.1 Problems

The core problem our thesis deals with is the insufficient matching result quality and support for scalability. This problem has also been illustrated in the previous example in Sect. 3.1.3. In the following we will describe our cause analysis and the resulting subproblems.

1. *Insufficient matching quality and scalability* The cause problem we identified corresponds to our research question that is the root of the problem hierarchy in Fig. 3.4. The two cause problems separate the problem tree into the problem of incorrect and incomplete results as well as insufficient support for industrial scale matching.

2. *Incorrect and incomplete matching results* The matching quality of today's matching systems in general and metamodel matching systems in particular is insufficient. That means the results of today's matching systems are partially incorrect and incomplete. Consequently, they leave room for improvement. This problem is backed by two arguments. First, we investigated the state of the art of metamodel matching and implemented a system incorporating these techniques. We

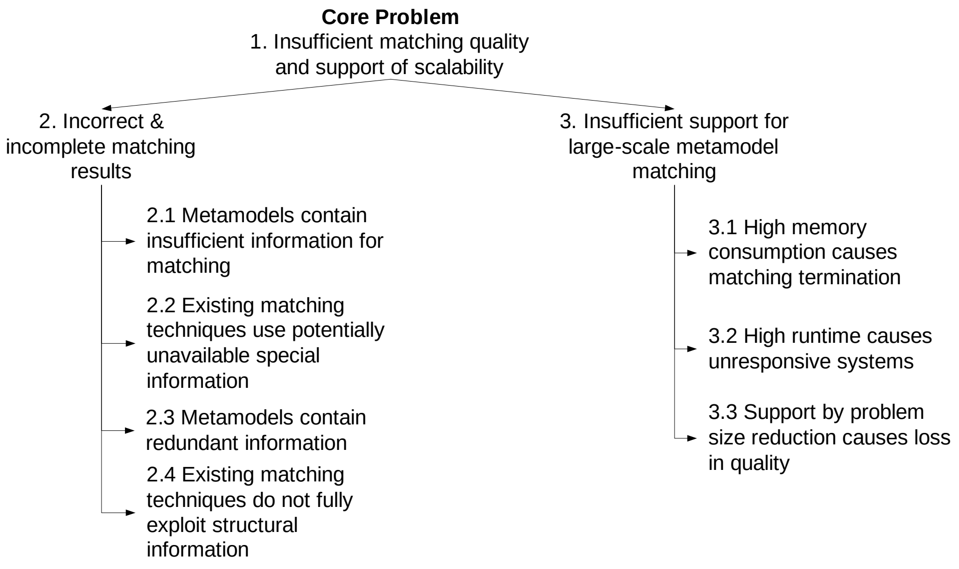

Core Problem
1. Insufficient matching quality and support of scalability
2. Incorrect & incomplete matching results
2.1 Metamodels contain insufficient information for matching
2.2 Existing matching techniques use potentially unavailable special information
2.3 Metamodels contain redundant information
2.4 Existing matching techniques do not fully exploit structural information
3. Insufficient support for large-scale metamodel matching
3.1 High memory consumption causes matching termination
3.2 High runtime causes unresponsive systems
3.3 Support by problem size reduction causes loss in quality

Figure 3.4: Problem hierarchy

carried out an analysis using real-world data and demonstrated that only half of all matches were found [151] confirmed for large-scale schemas by [125]. Second, a metamodel contains insufficient information for matching.

2.1 *A metamodel contains insufficient information for matching* This observation has already been made in schemas [126] and is easily transferred to metamodels. A metamodel (see Definition 1) only contains individuals and labels, thus static structure. It lacks not only information about executional semantics, but also about the context and other metadata, which is needed in order to formulate mappings. Sometimes, only a domain expert is capable of specifying a mapping, because of his knowledge of the semantics of the elements.

2.2 *Existing matching techniques use (unavailable) special information* There are matching techniques which try to make use of additional special information (metadata) or background knowledge [29]. However, this leads to the second problem of existing matching techniques using (unavailable) special information. The information they make use of does not exist per se. Therefore, one has to assume that this information is missing, thus their techniques may even decrease the result quality.

2.3 *Metamodels contain redundant information* Another possible cause is the problem that metamodels contain redundant information.

Even though it is considered bad style and should be avoided, real-world metamodels contain redundant information such as duplicated information or multiple ways of expressing the same entity. This redundant information is misleading for matching techniques and decreases the result quality.

2.4 *Existing matching techniques do not fully exploit structural information* The fourth problem identified by us is that existing matching techniques do not fully exploit structural information. The rationale behind this problem is that the more information available and the more matching techniques make use of it, the better the achieved result. For an overview of existing matching systems refer to Sect. 4.1.

3. *Insufficient support for large-scale metamodel matching* The problem of insufficient support for large-scale metamodel matching states that runtime and memory consumption problems arise when increasing the size of the metamodels to be matched to a certain extent. That means matching metamodels of arbitrary size is not supported. The main reason for this problem is the quadratic number of comparisons for matching. The matches are derived by comparing each element of one metamodel with each element of another metamodel, thus there is a quadratic number of comparisons due to the cartesian product of elements being compared.

   3.1 *High memory consumption causes matching termination* One consequence of large-scale metamodels and a quadratic number of comparisons is an increasing memory consumption, which finally leads to problems of insufficient memory. When hitting memory boundaries the process has to terminate and the resulting mapping is empty. Currently, metamodel matching systems are not able to match such metamodels.

   3.2 *High runtime causes unresponsive systems* Matching of large-scale metamodels naturally leads to an increasing runtime. In consequence, the response time is in the range of minutes or even hours, provoking an unresponsive system and waiting times for a user.

   3.3 *Support by problem size reduction causes loss in quality* Another cause for insufficient scalability support is a resulting trade-off between result quality and scalability. As stated before in the two subproblems concerning memory and runtime the scalability issue has to be tackled by reducing the problem size. However, if one follows a context reduction approach as the generic matching and differencing system EMF Compare [10] the matching is limited to neighbours of an element to match instead of considering the whole metamodel, as a consequence the result quality decreases considerably, since less information can be used for matching.

#### 3.2.1.2 Scope

We define the scope of our work by selecting some of the identified problems and neglecting the others based on the possibility to improve shortcomings by graph-based techniques. Furthermore, we want to pursue a generic solution, thus we neglect problems which require special support. We also base our scope on the restriction of a matching system independent solution, i. e. there is no need to change an existing system besides adding our proposed solution. The following problems will be tackled: metamodels contain redundant information (2.3), existing matching techniques do not fully exploit structural information (2.4), high memory consumption (3.1), high runtime (3.2), and the trade-off between scalability and quality (3.3). Consequently, we contribute to the solution of the problem of insufficient matching result quality due to incorrect and incomplete results (2.) and we aim to provide support for matching metamodels of large-scale (3.). We also contribute to the solution of the core problem of insufficient matching quality and scalability. The problems which we do not consider (2.1, 2.2) and which are out of scope will be discussed in the two subsequent paragraphs.

**2.1 Insufficient matching information in metamodels** This problem has been identified in the areas of ontologies [33] and XML schemas [126], and is easily transferred to metamodels. Attempts to solve the problem have been made by adding metainformation such as mappings to upper level ontologies [28], etc. However, none of these proposals could present a complete solution, i. e. none could identify all matches. Therefore, this problem will not be tackled by our work; instead we will concentrate on using already available information.

**2.2 Existing matching techniques use (unavailable) special information** We will not tackle this problem but derive our approach from it. Rather than finding a way to add the missing information we pursue the usage of already existing information, namely the graph structure of a metamodel. We decide to only make use of already available information, because we do not want to impose another task for the user. Adding the information would require an additional effort for the matching task. Furthermore, the additional special information requires an adaption of the infrastructure to support the loading, storing, etc. thereof.

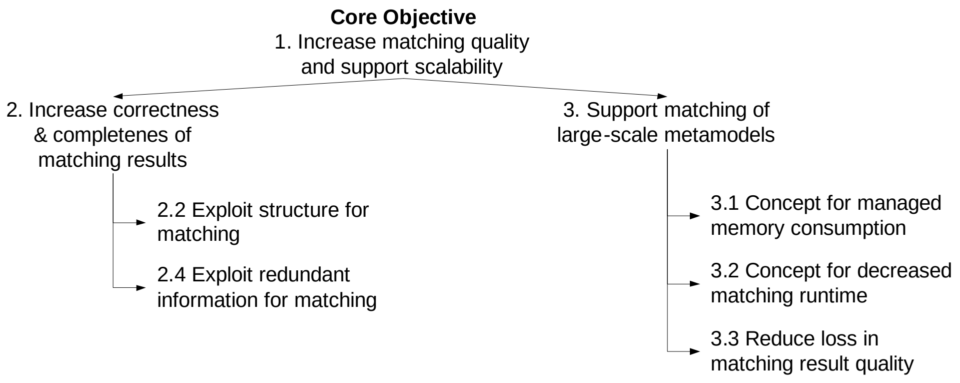

Core Objective
1. Increase matching quality and support scalability
2. Increase correctness & completenes of matching results
3. Support matching of large-scale metamodels
2.2 Exploit structure for matching
2.4 Exploit redundant information for matching
3.1 Concept for managed memory consumption
3.2 Concept for decreased matching runtime
3.3 Reduce loss in matching result quality

Figure 3.5: Objective hierarchy

### 3.2.2 Objectives

The objectives we define are derived from the problem hierarchy as in Fig. 3.4. Removing the problems 2.1 and 2.2 which are out of scope we present the objective hierarchy in Fig. 3.5. The hierarchy follows the example of the problem hierarchy with the final objective of increasing matching quality and support of scalability.

1. *Increase matching quality and support scalability* The two subobjectives, an increase in quality and concepts for a scalability support form our main goal.

2. *Increase correctness and completeness of matching results* The correctness and completeness define the quality of the matching results, if both are increased without decreasing each other, the overall result quality is improved. An improvement can be achieved by pursuing the following two subobjectives tackling the previous problems.

   2.2 *Exploit structural information for matching* The structural information, i. e. a metamodel's graph structure, should be used for matching. That means matching algorithms should use this information for matching calculation. There are several algorithms in graph theory that make use of the structure for similarity calculation, a detailed analysis of these algorithms is presented in Chap. 5.

   2.4 *Exploit redundant information for matching* Redundant information in the metamodels to be compared can cause misleading matches but can also be used for matching. This information should be facilitated to increase the result quality. Redundant information first needs to be identified by so-called mining algorithms and then be matched. In Chap. 5 we will present an analysis of the existing algorithms.

3. *Support matching metamodels of large-scale size* This objective implies concepts to tackle the problems of memory and runtime. Along with these two problems the trade-off between scalability and quality has to be considered.

   3.1 *Concept for managed memory consumption* Scalability should be supported by a concept which allows for managed memory consumption through the matching process. This memory consumption should also allow for matching of metamodels of arbitrary size.

   3.2 *Concept for reducing matching runtime* The complexity of matching is quadratic to the size of the input metamodels' elements to be matched. Consequently, the runtime increases quadratically with the increase of input elements. The objective is to develop a concept which reduces the matching runtime.

   3.3 *Optimize trade-off between scalability and matching result quality* While introducing a concept for managed memory consumption and reduced runtime the problem is split into smaller subproblems. This problem reduction implies a trade-off between result quality and scalability, which should be optimized. That means support for managed memory consumption while having a reasonable runtime and result quality.

### 3.2.3 Requirements

The objectives identified allow for a definition of requirements to find solutions that fulfil these objectives. The requirements are given in Tab. 3.2.3 and grouped into three groups, the first two deal with Objective 2.2 to increase correctness and completeness and the third group deals with the support of scalability, that is Objective 2.4.

R1 *Exploit structural and redundant information* The solution should automatically exploit available structural and redundant information, i. e. without user interaction.

  R1.1 *Exploit redundant information and graph structure* The solution should exploit the explicit graph structure of a metamodel. This information should be used for matching in combination with other matching techniques, e. g. linguistic ones. In addition, redundant information should be exploited for metamodel matching.

Table 3.1: Overview of requirements

| Requirement | Subrequirements |
|---|---|
| R1 Exploit structural and redundant information | R1.1 Exploit graph structure and redundant information<br>R1.2 Improve correctness and/or completeness<br>R1.3 Maximal quadratic complexity |
| R2 Matching metamodels of large scale | R2.1 Partition input metamodels<br>R2.2 Reduce runtime<br>R2.3 Minimal losses, preserve, or even improve result quality |

R1.2 *Improve correctness and completeness* While matching the redundant information the solution should improve correctness and completeness of the mappings calculated.

R1.3 *Maximum quadratic complexity* If possible, the solution should adhere to the quadratic complexity imposed by the matching problem.

R2 *Matching large-scale metamodels* The solution should allow matching metamodels of large-scale size.

R2.1 *Partition input metamodels* The solution should support a partitioning of the input metamodels which allows for a reduced problem space for matching.

R2.2 *Reduce runtime* The solution should reduce the overall runtime of the matching process.

R2.3 *Minimal loss of result quality* The solution should minimize the loss of result quality. If possible it should preserve or even improve the correctness and completeness of mappings calculated.

### 3.2.4 Approach

The requirements identified specify the solution to fulfil our objectives (cf. Sect. 3.2.2). The connection is the approach on developing a solution based on the requirements for the objectives. Figure 3.6 depicts our approach and its single steps.

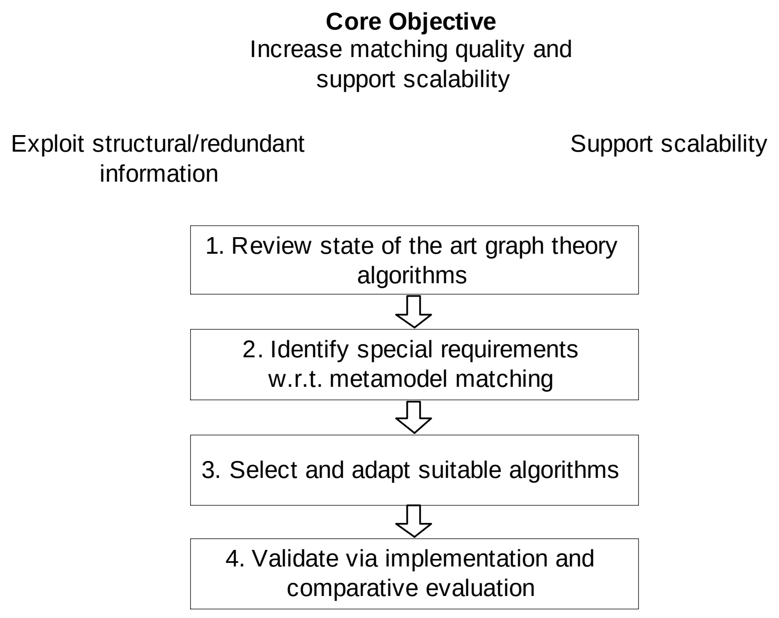

**Core Objective**
Increase matching quality and support scalability
Exploit structural/redundant information
Support scalability
1. Review state of the art graph theory algorithms
2. Identify special requirements w.r.t. metamodel matching
3. Select and adapt suitable algorithms
4. Validate via implementation and comparative evaluation

Figure 3.6: Steps of our problem solving approach on increasing quality and scalability

The basis is the main objective of increasing the matching result quality and the support for scalability. It is split into the two objectives concerned with structural and redundant information, and matching metamodels of arbitrary size (scalability). These three objectives will be approached by us in a uniform manner consisting of the following four steps:

1. *Review state of the art graph theory algorithms* The main focus of our work is on using structural (graph) information. Therefore, we first review generic algorithms in the field of graph theory which provide solutions for the respective objectives.

2. *Identify special requirements w.r.t. metamodel matching* The next step is to narrow down the set of algorithms reviewed. We identify requirements which are imposed by the domain of metamodel matching. That means requirements such as specific graph properties, graph type restrictions, complexity, etc. They are used as a basis for a selection and adaptation of the generic graph algorithms.

3. *Select and adapt suitable algorithms* The algorithms studied are compared w.r.t. the identified requirements. The algorithms which fulfil the requirements are adapted and changed according to the domain of metamodel matching.

4. *Validate via implementation and comparative evaluation* The algorithms designed in the selection and adaptation step are implemented in a prototype and finally evaluated in a comprehensive evaluation using real-world data sets. Finally, the results obtained are compared to the objectives and requirements.

### 3.2.5 Research question

The research question given and motivated in our introduction is: ”How can metamodel graph structure be used to improve the correctness, completeness, runtime and memory consumption of metamodel matching?”.

We defined three hypotheses H1, H2, and H3. H1 and H2 state that the subgraphisomorphism and graph-mining algorithms will improve the result quality. These hypotheses are supported by our requirements for matching R1 and also by our approach in surveying the state of the art in graph theory and selecting and adapting suitable algorithms. Consequently, this will contribute to the answer of our research question.

The hypothesis H3 deals with planar partitioning for large-scale matching and is supported by the requirements of R2 and again by our approach. In investigating the state of the art in graph partitioning and clustering we derive the planar partitioning and thus contribute to H3 and finally to our research question.

## 3.3 Summary

In this chapter we gave a motivating example for metamodel matching and challenges regarding matching quality and scalability. Moreover, we performed a problem analysis, derived objectives and requirements, and outlined our approach on a solution for the objectives.

**Motivating example** The motivating example presented by us originates from the area of retail stores. A retail store provides a system for cashiers and goods movement provided by a third-party where all data is gathered in a central ERP system. The third-party systems have to be integrated with the central system provided by SAP. Thereby, metamodel matching assists in specifying mappings for this data integration problem. However, the result quality suffers from missing information, different naming conventions, etc. Also both metamodels consist of more than 900 elements, which leads to runtime and memory consumption issues.

**Problem analysis** We performed a problem analysis on the matching result quality and scalability issues based on ZOPP. Thereby, we first identified two cause problems: (1) insufficient correctness and completeness of matching results and (2) insufficient support for large-scale metamodel matching. We also performed a root-cause analysis for these two problems. This was followed by a scope identification and a derivation of the objectives which constitute a solution to the cause problems. Namely the objectives are: exploit structural information for matching, exploit redundant information for matching, and support matching of metamodels of arbitrary size.

These objectives lead to a set of requirements our solution has to fulfil, which are refined in the respective chapters 5 and 6 where our solution is presented. Finally, we presented our four steps for solving the cause problems, which are pursued for all three objectives. They involve a study of existing graph algorithms based on an objective specific requirement analysis. Based on the requirements, algorithms are selected and adapted in the context of large-scale metamodel matching. The proposed solution is then investigated in a comprehensive evaluation.

# Chapter 4

# Related Work

Having set the foundation of our work and analyzed the problems of matching quality and scalability, we want to give an overview on state of the art matching systems. We first introduce the areas of schema, ontology, and metamodel matching and selected systems. Subsequently, we present related structural matching approaches w.r.t. our proposed planar graph edit distance and graph mining-based matching. We point out the shortcomings of the existing approaches and how our proposals may complement them, thus establishing our research ground. This is also done for large-scale matching approaches, discussing their short-comings in either quality or memory consumption and their relation to our planar partitioning.

## 4.1 Matching Systems

Several systems have been developed in the areas of schema, ontology, and metamodel matching. Although all systems tackle the problem of meta data matching, they were and are being researched in a relatively independent manner.

Prior to studying matching systems, one needs to clarify the relation of the domains of schemas, ontologies, and metamodels. In this work, we adopt the perspective of Aßmann et al. [6] and extended their perspective by including XML schemas. Schemas, ontologies, and metamodels provide vocabulary for a language and define validity rules for the elements of the language. The difference is in the nature of the language, either it is prescriptive or descriptive [6]. Thereby, schemas and metamodels are restrictive specifications, i. e. they specify and restrict a domain in a data model and systems specification, hence they are prescriptive. As a complement, ontologies are descriptive specifications and as such focus on the description of the environment. Schemas are used for a tree-based description of data for processing by a machine, whereas metamodels abstract a domain of interest in a graph-based structure. Therefore, using a similar vocabulary made of linguistic and structural information, the three domains differ in their purpose.

Since the three domains have a different purpose, matching systems for these domains have been developed independently. First, (1) schema matching systems have been developed to mainly support business and data integration, as well as schema evolution [24, 77, 100, 107, 126]. Thereby, the schema matching systems take advantage of the explicit tree structure. Second, with the advent of the Semantic Web, (2) ontology matching systems are dedicated to ontology evolution and merging, as well as semantic web service composition and matchmaking [17, 53, 76, 154, 68, 160]. They are especially of use in the biological domain for aligning large taxonomies as well as for classification tasks. Finally, in the context of MDA, which requires refinement and model transformation (3), metamodel matching systems are concerned with model transformation development, with the purpose of data integration as well as metamodel evolution [10, 37, 38, 73, 151].

In the following we will describe matching systems from each of these domains and their applied matching techniques to provide an overview on the state of the art in matching.

### 4.1.1 Schema matching

Based on a survey of schema-based matching approaches [138] three widely-known systems applicable in the schema matching domain were selected as representatives of the group. The three selected systems are COMA++ [24], Cupid [100], and the similarity flooding algorithm [107]. Similarity flooding was first implemented for schema matching, but has since been adapted and is nowadays used in systems in other domains, e. g. ontologies [160] and metamodels [38].

**Similarity flooding** The similarity flooding algorithm [107] is a hybrid matching algorithm that relies on the hypothesis that if two entities are similar then there is also a certain similarity between their neighbours. Schemas are represented as directed labeled graphs and the algorithm runs in an iterative fix-point computation to produce a mapping between vertices of input graphs. First, initial string-based techniques are applied to obtain starting similarity values for the fix-point computation. Starting from similar elements, the similarity is propagated to neighbour elements through propagation coefficients. The process is finished when a fix point is reached and the final mapping is then returned. Similarity flooding has been integrated with linguistic techniques in the protoype model management system Rondo [108].

**COMA++** COMA++ [23] is a matching system that uses a COmbination of MAtching techniques to produce mapping results. The matchers operate on an internal model, namely a directed acyclic graph. Schema elements are represented by nodes in the graph and they can be connected by directed links representing different types of relationships. The matcher library consists of linguistic matchers, matchers that combine structural and terminological matching techniques, as well as matchers based on a repository of structures. The results from the different matchers are aggregated and selected to a single similarity value for a pair of elements. It has to be noted that COMA++ was extended to also be applicable in the ontology domain.

**Cupid** Cupid [100] is a matching system that applies both terminological and structural matchers. Input schemas are imported into an internal representation as trees, on which the matching techniques operate. The matching algorithm has three phases and operates only on tree structures to which non-tree cases are reduced. The first phase applies string-, language-, and constraint-based techniques, as well as linguistic resources to compute similarity. During the second phase local tree-based techniques are applied. In the third phase a weighted aggregation and threshold-based selection produce the final mapping.

A comparison of our concepts with the approaches of COMA++ and similarity flooding can be found in Sect. 4.2 and 4.3.

### 4.1.2 Ontology matching

The ontology matching domain is also related to our work and presents a multitude of systems (more than 30). Therefore, we selected systems based on their performance in the Ontology Alignment Evaluation Initiative (OAEI) contest benchmark[1] and their relatedness to our approach, i. e. they are considered if they apply graph-based techniques or tackle the scalability problem. Three systems were chosen, namely Aflood [53], Falcon-AO [69], and RiMOM [160].

**Aflood** The Aflood (Anchor-Flood) system [53] targets the mapping of large scale ontologies. It does not compare each element of two ontologies, but rather chooses so called anchor-elements to group elements based on an anchor's neighbourhood. Elements are assigned to a group based on the inheritance relation. The elements in the resulting groups are matched using local matchers. The local matchers apply string-based and language-based techniques, and make use of linguistic resources. Additionally, instance matching is supported as part of the system. Based on this infor-

[1] `http://oaei.ontologymatching.org/`

mation a structural similarity among elements in a so called semantic link cloud [53] is computed to produce the alignments (mappings).

**Falcon-AO** Falcon-AO aims at automatic ontology mapping by a matcher library of four matchers. The library consists of two linguistic based matchers, one calculates element similarity based on a vector space model and the other is a string edit distance. The third matcher is called GMO (Graph Matching for Ontologies) [63], which is a graph based matcher. The last matcher represents the partitioning of large ontologies and GMO and the partitioning matcher are discussed in Sect. 4.2. Finally, the values from the matchers are aggregated using a weighted approach.

**RiMOM** The Risk Minimization Ontology Mapping system (RiMOM) [95, 160] is an ontology matching system that uses several matching configurations. The system evaluates preliminary information, i. e. input metrics, about how different configurations would perform and prefers some of them based on that information. The system applies linguistic matching techniques, calculating a string edit distance, the linguistic resource WordNet, a vector-based similarity, and path similarity. Hence, it uses linguistic and local matching based on trees. The different similarity values are combined to produce a single value to give the final mapping.

### 4.1.3 Metamodel matching

Although the research area of metamodel matching is developing, it does not yet offer as many matching systems as the ontology and schema matching domains. It has to be noted that we consider only metamodel matching systems, and no model matching tools as described in [81]. Six metamodel matching systems were identified by us, namely the Atlas Model Weaver [37] (extended by AML [45]), EMFCompare [10], GUMM [38], MatchBox [151], which is the system proposed by us and described in Sect. 7.2, ModelCVS [73] (implicitly applying COMA++ from the schema matching domain), and SAMT4MDE [18].

**AMW** Fabro et al. [37] proposed the Atlas Model Weaver (AMW), a system for semi-automatic model integration using matching transformations and weaving models. A weaving model captures the different possible relationships between two models. The matching is performed by creating the weaving model between the two input metamodels. Thereby, both linguistic and structural information is exploited. The linguistic approach calculates string similarity between names of elements and also uses dictionaries where the structure similarity computation follows the similarity flooding approach. In

AML [45] an extension was proposed which allows the modelling of domain-specific matchers to improve the matching results for specific matching tasks, but with the drawback of low performance [46].

**EMFCompare** The main focus of EMFCompare [10] is to calculate differences between versions of the same model by applying both matching and differencing techniques. However, the authors of EMFCompare explicitly state its applicability for metamodels, thus we also added it because of its popularity. The matching engine uses statistical information, heuristics and instance information. The name of an element, its content, type and relations are the metrics being analyzed in order to produce a similarity value. Each of the aforementioned information is represented as a string and a string edit distance is applied for similarity calculation. In order to reduce information noise and to provide better results, EMFCompare applies a filter on the match results.

**GUMM** Falleri et al. proposed the Generic and Useful Model Matcher system (GUMM) [38] for metamodel matching based on the similarity flooding algorithm [107]. Similar to us they use directed labeled graphs to internally represent metamodels. On these graphs they apply the similarity flooding algorithm. For the initial similarities, which the algorithm needs, string-based metrics are used, including for example the Levenshtein Distance [153]. A fix-point iteration process is then started to produce the result mapping.

**ModelCVS** The ModelCVS system [73] proposes an approach to match metamodels, in which metamodels are semi-automatically transformed into ontologies, a process which the authors call metamodel lifting. Afterwards an alignment (matching) between the ontologies is performed and finally a bridging between the metamodels is derived from the ontology mapping. To find similarities between the resulting ontologies, several tools were applied and the one with the best results was chosen, namely, COMA++ [24].

**SAMT4MDE** The Semi-Automatic Matching Tool for Model Driven Engineering (SAMT4MDE) [18] is a tool for metamodel matching. A matching algorithm based on cross-kind-relationships is used and extended to also use structural information from the models being matched. The overall similarity value between two classes is a weighted sum of the base similarity and the structural similarity. The base similarity uses the idea of cross-kind-relationship implication, for example if A contains B and B is-a C, then A contains C. The base similarity calculation is also based on a repository of taxonomies. The structural similarity is computed using taxonomy- and tree-based techniques.

## 4.2 Matching Quality

The majority of matching systems, including schema, ontology, and metamodel matching, applies local matching techniques [100, 107, 23, 95, 53, 37, 10, 38, 18]. As described before, local matching does not take the complete structure into account. Moreover, matching in most of the systems is restricted to trees. However, there are approaches and systems which apply global graph-based techniques and which are consequently related to our work for concepts on a graph edit distance (Sect. 5.1). We discuss those approaches in the following section, highlighting the difference to our approach. Next, we describe related approaches that make use of region-based (redundant) information for matching, which is related to the graph mining-based matching as proposed by us in Sect. 5.2.

**Graph matching** In metamodel matching the only graph matching technique used in systems [36, 38] is similarity flooding [107], which originates from schema matching. This local technique suffers from two main drawbacks, first it does not consider the graph structure as a whole but rather investigates neighbour relations. Second, it uses propagation and thus heavily relies on input matches, i. e. linguistic information, which may propagate an error multiple times to wrong results. In contrast, our approach uses the whole graph structure and does not suffer from error propagation. It is worth noting, that the results of our planar graph edit distance may serve as an input to similarity flooding. However, this option has not been investigated by us, since a parallel similarity combination tends to be superior to similarity flooding as demonstrated by [23].

In the field of ontology matching two approaches for global graph-based matching were proposed. The first approach is GMO [63], which is part of the Falcon system. Its basic idea is to transform an input ontology into a bipartite graph. Inside a bipartite graph the vertices can be split into two disjoint sets. Between each element of one set and each element of the other set an edge is established. These edges represent the similarity. GMO uses structural measurements to iteratively refine the matrix. Finally, after the iterations converge a mapping can be derived. The cause problem of the approach is that metamodels are not bipartite, and thus GMO cannot be applied. However, one could add vertices and edges to a metamodel until it becomes bipartite, but this procedure could lead to three-times the edges as also shown for ontologies in [56]. This is a fact, because all elements (vertices) have to be arranged into the disjoint groups. This added information would potentially produce misleading or wrong mappings, thus we neglect this approach.

Another interesting idea in ontology matching was proposed by [27], which deals with Expectation Maximization. The approach is to transfer an ontology into a direct labeled graph and interpret the problem of matching as maximum likelihood search. This search is known from the field of statistics and tries to evaluate the probability that a given match is valid for two given ontology graphs, thereby they make use of linguistic techniques. The computation is done over several iterations, each refining the previous result. The main drawback of the approach is its lack of scalability. The authors report a calculation time of 18 minutes for 40 elements [27], which makes it not applicable for real-world scenarios, which show on average 180 elements as given by our evaluation data in Sect. 7.3.

The work most related to our approach has been proposed in [162]. They propose a tree edit distance, which uses a general tree edit distance specialised for schema matching using a schema-specific cost function. In contrast our approach is not limited to trees and can be applied to any graph.

**Redundant information** The usage of redundant information has not been tackled explicitly so far. However, there are two non-metamodel approaches which are to some extent related to our work. The first one originates from ontology matching and aims at matchings calculated based on pre-defined patterns [130]. The authors define four patterns which are evaluated on a given ontology and based on the embeddings (occurences) mappings are derived. In contrast, our approach is capable of detecting arbitrary patterns which makes it generic, where their approach is ontology- and domain-specific.

Another related approach is presented by COMA++ [25] with the fragment approach. The idea is to compare the path of elements in a schema to differentiate between the context of re-occuring elements. The approach collects all multiple matches of a matching technique and refines them using their path. We follow a different approach and do not rely on a first matching but rather extract redundant information explicitly. Furthermore, with our Design Pattern matcher we are capable of identifying design patterns, which tend not to be present as multiple matches.

Table 4.2 gives an overview on related structural matching approaches. The approaches are classified according to whether they operate on graphs, perform local, global, or region-based matching, and if they are applicable to metamodels. Thereby, local, global, and region-based matching refers to our extended matching technique classification in Sect. 2.1.2.2. We conclude that our work is the only one applying global graph-based techniques on metamodels and also the only one applying region graph-based techniques on metamodel graphs.

Table 4.1: Overview on related structural matching approaches

| | Graph | Local | Global | Region | Meta-model |
|---|---|---|---|---|---|
| Our work | ✓ | ✓ | ✓ | ✓ | ✓ |
| Melnik et al. [107] | ✓ | ✓ | ✗ | ✗ | ✗ |
| Falleri et al. [38] | ✓ | ✓ | ✗ | ✗ | ✓ |
| Fabro et al. [37] | ✓ | ✓ | ✗ | ✗ | ✓ |
| GMO [63] | ✓ | ✗ | ✓ | ✗ | ✗ |
| Doshi & Thomas [27] | ✓ | ✗ | ✓ | ✗ | ✗ |
| Do et al. [25] | ✗ | ✓ | ✗ | ✓ | ✗ |
| Ritze et al. [130] | ✓ | ✗ | ✗ | ✓ | ✗ |
| Zhang [162] | ✗ | ✗ | ✓ | ✗ | ✗ |

## 4.3 Matching Scalability

In this section we present related approaches to scalability support of matching systems. Thereby, we again consider schema, ontology, and metamodel matching, because in metamodel matching only EMFCompare [10] takes scalability into account. Large-scale matching follows one of the following approaches:

1. Light-weight matching,
2. Context reduction,
3. Self-configuring matching workflows, or
4. Partitioning.

**Light-weight matching** Light-weight matching employs only simple matching techniques, such as linguistic ones, to save runtime and memory. Still those approaches do not solve the runtime and memory consumption problem. They only postpone it, because they still need to match the cartesian product of source and target elements. In metamodel matching EMFCompare [10] and in ontology matching the RIMOM [95] system follow this approach.

**Context reduction** Another proposal for scalability support in ontology matching is given by Aflood [54]. They apply an incremental approach starting to match only certain elements, so-called anchors. If matches are found, the neighbours of those anchors are matched recursively until no match can found. Although the approach seems to perform well on certain data, in the worst case, e. g. identical metamodels, again the complete data is matched. Consequently, the scalability problem is only tackled partially.

**Self-configuring matching workflows** Peukert [123] and RIMOM [95] apply a different approach by making use of self-configuring matching workflows. Thereby, light-weight matching techniques serve as a selector of elements to be matched, to reduce the number of comparisons to be made for subsequent techniques. Unfortunatly, this also only tackles runtime but not memory and hence does not solve the scalability problem.

**Partitioning** Partitioning is an approach following the divide and conquer paradigm. Thereby, the input is separated into parts and those parts are either used for selection of match partners or for an independent matching of the parts. COMA++ [25] follows a partitioning approach by identifying fragments. Those fragments represent either types or user-defined patterns, and determine the comparisons to be made. This approach reduces the number of comparisons for matching but does not tackle the memory problem, because the matchers still use the complete metamodel (schema) for matching.

Falcon-AO [156] took this idea and applied a partitioning based on the graph clustering algorithm ROCK [156]. To reduce the number of comparisons for matching, they apply a threshold-based assignment, as also studied by us. However, the cause problem of this approach is the partition similarity calculation. FALCON-AO proposes to calculate anchor pairs (source-target pairs with a high similarity) using name and annotation matchers prior to partitioning or being defined by a user. The matching is only applied to partitions with shared anchors. Consequently, both ontologies have to be matched completely with light-weight techniques which again imposes memory issues.

Another approach based on the prominent clustering algorithm k-means also applies partitioning for schemas [140]. Thereby, they assume a user-defined small-sized pattern which helps in selecting the centroids (similar to the anchors). All elements with a neighbourhood larger than the pattern are considered as candidates. In the next step all partitions consisting of centroids are incrementally extended until a fix-point is reached, i. e. no element gets assigned to a partition. The authors note that the algorithm sometimes produces unbalanced clusters, which negatively influences the solution. Again, the clustering algorithm reduces runtime and memory in most cases, but it cannot ensure a maximal cluster (partition) size and thus does not tackle the memory problem in general.

Recently, Gross [50] proposed a partitioning for COMA++. The approach is to sequentially partition an input ontology into maximal-sized partitions. These partitions are matched independently and in parallel. To tackle the

Table 4.2: Overview on related large-scale matching systems

| | Run-time | Memory | Partit-ioning | Struct. Pres. | Assign-ment | Meta-model |
|---|---|---|---|---|---|---|
| Our work | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| COMA++ Part. [50] | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ |
| Falcon-AO [156] | ✓ | ✗ | ✓ | ✓ | ✓ | ✗ |
| COMA++ Frag. [25] | ✓ | ✗ | ✓ | ✗ | ✗ | ✗ |
| k-Means [140] | ✓ | ✗ | ✓ | ✗ | ✗ | ✗ |
| Aflood [54] | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ |
| EMFCompare [10] | ✓ | ✗ | ✗ | ✗ | ✗ | ✓ |
| RIMOM [95] | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ |
| Peukert [123] | ✓ | ✗ | ✗ | ✓ | ✗ | ✗ |

memory problem local information is stored in elements by pre-processing. That means that an element's children, siblings, parents and name similarity are pre-computed and stored for parallel matching. Besides imposing a matching overhead in pre-processing, the authors also note the major drawback of their approach; it does not allow for global structural matching or similarity flooding. The reason is the missing structural information since it cannot be stored without storing the complete ontology, which again does not solve the problem. In contrast, our approach allows for a structure-preserving partitioning and thus for global graph-based matching and similarity flooding.

Table 4.3 summarizes our related work on large-scale matching. The features investigated were runtime and memory. We discuss whether the corresponding approach reduces the runtime and memory and thus tackles the problem. Partitioning states whether the corresponding approach applies a splitting into parts, while structure-preserving investigates whether the approach still allows for global structural matchers. The assignment column gives insights into whether the approach utilizes an assignment of partitions and the last column specifies the applicability to metamodels.

To conclude, our approach differs from the state of the art in tackling both memory and runtime while providing a structure preserving partitioning and explicit consideration of assignment.

## 4.4 Summary

Our work is related to the fields of schema, ontology, and metamodel matching. Therefore, we gave an overview on matching systems from these related areas and discussed related work dealing with improvements of matching quality by structural matching and matching scalability in detail.

**Matching quality** The common approach on graph matching across all three domains [37, 38, 160] is similarity flooding [107]. However, this algorithm is a similarity propagation approach which does not use the global graph structure. Two approaches from ontology matching either suffer from the assumption of bipartite graphs [63] or scalability issues [27]. A related approach from the area of schema matching calculates the edit distance between trees [162], where we calculate an edit distance for graphs (Sect. 5.1).

Redundant information is not explicitly used for matching. However, the fragment-based approach of COMA++ [25] tries to identify complex types in schemas to derive a similarity between those fragments, but this approach relies on multiple matches. Another approach from ontology matching aims at calculating occurences of four pre-defined patterns [130], which we exceed by calculating these patterns via mining and then checking their occurences for matching as given in Sect. 5.2.

**Matching scalability** Approaches on matching scalability can be separated into light-weight matching [10], context reduction [54], self-tuning matching processes [95, 123], and partitioning [140, 25, 156, 50]. Although all approaches tackle the runtime issue one way or the other, the memory consumption problem is only tackled by [156, 50]. Thereby, [156] suffers from its domain specifics and partition assignment calculation, which may lead to memory issues. The alternative approach [50] on the other hand does not apply a structure-preserving partitioning and is therefore not applicable to global graph-based algorithms or similarity flooding. These issues are tackled by our planar partitioning as given in Chap. 6.

# Chapter 5

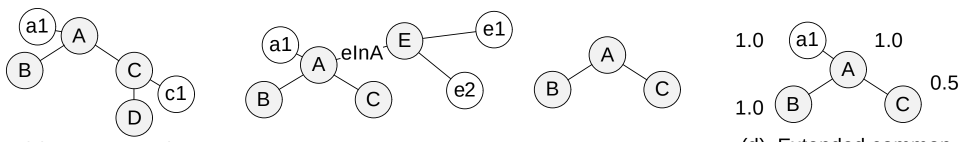
# Structural Graph Edit Distance and Graph Mining Matcher

The problem of insufficient matching quality identified by us led to the objective of exploiting structural and redundant information. In this chapter we propose solutions for this objective by building on established graph theory. The solutions are three novel matching approaches: (1) a planar graph edit distance matcher, (2) a design pattern matcher, and (3) a redundancy matcher. These matchers are described by us in a uniform manner while categorizing them into graph matching and graph mining.

Since the algorithms perform structural graph-based matching we first conduct a requirements driven analysis of generic algorithms from graph theory. We then select algorithms as basis for our matchers and describe them. Subsequently, we provide an illustrative example calculation and remarks on our algorithms.

## 5.1 Planar Graph Edit Distance Matcher

In order to exploit structural information for increased matching quality we propose to apply an approximate subgraph isomorphism algorithm. Graph isomorphism algorithms have not been used for matching so far because of their intrinsic complexity especially in runtime [150]. However, our approach is based on an efficient, i. e. quadratic complexity, generic graph matching algorithm presented by Neuhaus and Bunke [113]. Their algorithm has been proposed for fingerprint classification and operates on unlabelled, planar graphs with each vertex having more than one edge. Since their algorithm shows good quality and a quadratic runtime we propose to adjust and extend the algorithm for metamodel matching. We propose techniques to consider labels and a pre-processing planarization. We also propose to consider one-edge vertices, e. g. attributes, for the edit distance calculation and thus for matching. To further increase the quality of the al-



(a) Source graph (b) Target graph (c) Common subgraph (d) Extended common subgraph

Figure 5.1: Example of source and target graphs, a common subgraph, and our extensions

gorithm we present our $k$-max degree approach based on partial user-input mappings.

Figure 5.1 depicts a source (a) and target graph (b), an example output of the original algorithm (c), and an example output of our extensions (d). The original output determines the common subgraph (*A, B, C*) as interpretation of the GED, omitting the vertex *a1*, because it violates its prerequisite of biconnectivity. Our extensions add a similarity value in [0,1] for each vertex as shown in Fig. 5.1 (d). Further, we include all one-edge vertices in calculation thus adding *a1* to the output graph.

In the following, we will first present an analysis of existing generic graph matching algorithms to justify our selection of the approximate algorithm by Neuhaus and Bunke. Subsequently, we introduce our algorithm for metamodel matching and an example calculation for the planar graph edit distance. Finally, we present an improvement in result quality by our $k$-max degree approach, which uses partial user-defined mappings as input. Since the seed mappings are based on selected elements presented to a user, we propose a simple, scalable, and efficient heuristic for the selection of these elements.

### 5.1.1 Analysis of graph matching algorithms

In the following we first perform our requirements analysis of graph matching algorithms. Second, we present suitable graph matching algorithms separating them into exact and inexact algorithms. Finally, we compare them to our criteria, thus establishing the justification and basis for our planar graph edit distance matcher.

#### 5.1.1.1 Requirements for metamodel matching

Our requirements analysis is based on the objectives defined in our problem analysis in Sect. 3.2.3. Recapitulating, we defined that a matching algorithm should (R1.1) exploit graph structure, (R1.2) improve correctness and completeness of existing matching systems, and (R1.3) have at most quadratic complexity.

When exploiting the graph structure an algorithm should support directed labelled graphs, i. e. metamodel graphs. While supporting metamodel graphs an algorithm should make maximum use of such structural information. That means a complete algorithm computing all possible matches should be preferred over an approximate algorithm. This is backed by the second requirement of improving the correctness and completeness, which is naturally more likely by a complete algorithm rather than an approximate one. The last requirement of quadratic runtime is applied on graph matching algorithms to preserve the upper bound of the overall matching process. To summarize, the requirements for graph matching algorithms in the context of metamodel matching are:

1. Support of directed labelled graphs
2. Maximal use of structural information
3. Quadratic complexity

These requirements will be used in the following section for an overview and comparison of graph matching algorithms.

#### 5.1.1.2 Overview of graph matching algorithms

Table 5.1 depicts an overview of the graph algorithms grouped into exact and inexact matching algorithms. These algorithms are arranged according to our requirements of temporal and spatial complexity, input graph restrictions and whether they are approximate or complete algorithms. The selection of particular algorithms is based on their popularity and their complexity, which should be quadratic if it is no standard algorithm.

To cope with larger graphs, algorithms have been proposed which either restrict the input graphs on certain properties or approximate the solution. Table 5.1 shows in the first five rows an overview of exact algorithms. The algorithm by Ullmann [144] exhibits an exponential complexity and is therefore not applicable for metamodel matching, since it cannot be used for mid-size structures (more than 20 elements). The approximate algorithm by Cordella [16] shows a better behaviour, but has an inacceptable worst case complexity. Luks [99] proposed an approach by restricting the input graph on each vertex having a bounded valence (degree as in number of neighbours). In metamodels this is not given, therefore we neglect this approach, because enforcing the property would result in a huge loss of information. A restriction of the input graphs on unique labels has been proposed by Dickinson [19]. Again metamodels do not show unique node labels, i. e. a fixed set of labels, thus we cannot apply the algorithm. Hopcroft [62] restricts the input on planar graphs proposing a polynomial algorithm, but due to its worse than quadratic memory complexity, we cannot apply this algorithm for metamodel matching.

The second group of rows in Tab. 5.1 shows three inexact algorithms which satisfy a polynomial execution time. The graph edit distance calculation proposed by Neuhaus [113] has a quadratic runtime and linear memory consumption by restricting the input to planar graphs. The approximate approach proposed by Riesen et al. [128] requires bipartite graphs as input and shows a quadratic runtime. Finally, Zhang [159] proposed an edit distance based on a tree representation showing a quadratic complexity both in runtime and space, where spatial complexity can be considered the same as memory consumption.

| Algorithm | Time Comp. | Spatial Comp. | Restriction | Nature |
|---|---|---|---|---|
| **Exact** | | | | |
| Ullmann [144] | $O(n!n^2)$ | $O(n^3)$ | – | complete |
| Cordella [16] | $O(n!n)$ | $O(n)$ | – | approx. |
| Luks [99] | $O(n^5)$ | n.a. | bounded valence | complete |
| Dickinson[19] | $O(n^2)$ | n.a. | unique node labels | complete |
| Hopcroft [62] | $O(n^2)$ | n.a. | planar graphs | complete |
| **Inexact** | | | | |
| *Neuhaus* [113] | $O(n^2)$ | $O(n)$ | planar graphs | complete |
| Riesen [128] | $O(n^2)$ | $O(n)$ | bipartite graphs | approx. |
| Zhang [159] | $O(n^2)$ | $O(n^2)$ | trees | complete |

Table 5.1: Overview and comparison of graph matching algorithms; n.a. – information is not available, $n$ – number of nodes

#### 5.1.1.3 Summary

Based on the comparison of matching algorithms in Tab. 5.1 we selected the algorithm we want to apply in the context of metamodel matching. Recapitulating the requirements, every algorithm supports directed labelled graphs. The second requirement states a maximal use of structural information, i. e. complete algorithms are preferred over approximate ones, therefore [16, 128] are neglected. Additionally, the tree-based approach [159] is not used, because a tree representation of a graph naturally contains less information than the graph itself. The third requirement is a maximal quadratic runtime, as a result [99, 144] are not considered by us. [99] requires unique node labels, which is not fulfilled by metamodel graphs; therefore we do not consider it. The standard algorithm by Hopcroft and Tarjan [62] shows memory issues and is consequently not considered. Finally, we selected [113] as the inexact algorithm for planar edit distance calculation in the context of metamodel matching, since it fulfils all of our requirements.

### 5.1.2 Planar graph edit distance algorithm

The approximate algorithm we propose to use for metamodel matching computes a lower bounded Graph Edit Distance (GED), instead of the minimal GED for two given input graphs. Thereby, the algorithm assumes planar labelled biconnected graphs, thus the input of the planar graph edit distance algorithm is a planar labelled biconnected source and target graph. The output of the original algorithm is a common subgraph and the corresponding GED. The output in terms of similarity is therefore one or zero. It is one if a vertex (element) is part of the common subgraph or zero if not. The algorithm calculates the minimal GED to reach a nearly maximal common subgraph. There are two parameters which influence the GED calculation: the costs for deletion and insertion as well as the function calculating the distance between two vertices. Finally, a seed match is needed to denote the starting point for calculation. Algorithm 5.1 presents the algorithm in pseudo code, which consists of:

1. Fetch the seed match (or the next match)
2. Calculate neighbourhood distance to identify new matches
   (a) Distance function based on linguistic and structural similarity
   (b) Processing of neighbours' attributes
3. Add new matches for next iteration and continue with step 2

**1. Seed match** In order to apply the algorithm in the context of metamodels, several adjustments and extensions are necessary. First, a seed match needs to be given which is used as a starting point for the neighbourhood edit distance calculation. Therefore, we define the root package vertices match as input.

**2. Neighbourhood distance** Starting with the seed match $v_s^0 \rightarrow v_t^0$ (both metamodels' root packages) the neighbours of the source vertex N($v_s$) and the neighbours of the target vertex N($v_t$) are matched against each other. The results are matches between a set of source and target vertices, this serves as input for the next iteration.

The match between both neighbourhoods (N($v_s$), N($v_t$)) is calculated, using a matrix of all N-vertices' distances, which are searched for the minimal edit distances. Thereby, one takes advantage of the planar embedding, which ensures that the neighbourhood of $v_s, v_t$ is ordered. So, there is no

**Algorithm 5.1** Planar graph edit distance algorithm

```
1    input: $G_s$, $G_t$, $v_s^0 \rightarrow v_t^0$
2    variable:  FIFO queue Q
3    add seed match $v_s^0 \rightarrow v_t^0$ to Q
4    fetch next match  $v_s \rightarrow v_t$ from Q
5      match neighbourhood N($v_s$) to the neighbourhood N($v_t$)
6      add new matches occurring in step 5 to Q
7    if Q is not empty, go to step 4
8    delete all unprocessed vertices and edges in $G_s$ and $G_t$
9    return output mapping
```

need to calculate all distances for all permutations of all vertices of N($v$) by applying any cyclic edit distance approach. Since the algorithm only needs to traverse the whole graph once, and the cyclic string edit distance based on the planar embedding has a complexity of $O(d^2 \cdot \log d)$ [122], the overall complexity is $O(n \cdot d^2 \cdot \log d)$ ($n$ the number of vertices and $d$ is the number of edges per node). The vertex distance function used is based on structural and linguistic similarity, with a dynamic parameter calculation for weights of the distance function.

**2. (a) Distance function** We defined the distance function of a vertex as a weighting of structural and linguistic similarity, since this allows us to consider structural and linguistic information and compensates for the absence of one of them. The linguistic similarity $ling(v_s, v_t)$ is the average string edit distance between the vertex and egde labels. Thereby, we make use of the Levenshtein distance of the name matcher of MatchBox as given in Sect. 7.2.

$$dist(v_s, v_t) = \alpha \cdot struct(v_s, v_t) + \beta \cdot ling(v_s, v_t) \wedge \alpha + \beta = 1 \tag{5.1}$$

The structural similarity $struct(v_s, v_t)$ is defined as the ratio of structural information, i. e. the attribute count ratio $attr(v_s, v_t)$ and reference count ratio $ref(v_s, v_t)$, thus allowing for a fast computation. The parameters do not have to be necessarily the same as for the overall distance.

$$struct(v_s, v_t) = \alpha \cdot attr(v_s, v_t) + \beta \cdot ref(v_s, v_t) \wedge \alpha + \beta = 1 \tag{5.2}$$

The ratio between the attributes and references, i. e. the references of a source vertex $v_s$ to the references of a target vertex $v_t$, are defined as the attribute and reference count ratios respectively.

$$attr(v_s, v_t) = 1 - \frac{|count_{attr}(v_s) - count_{attr}(v_t)|}{max(count_{attr}(v_s), count_{attr}(v_t))} \tag{5.3}$$

Finally, we defined the cost for insertion and deletion as being equal, since both operations are equal for similarity calculation. Consequently, a vertex to be matched has to be deleted if no matching partner with a similarity greater than $T$ can be found. The following threshold definition must hold to keep a vertex $v_s$:

$$sim(v_s, v_t) = 1 - dist(v_s, v_t) \geq\ 1 - cost_{del}(v_s) - cost_{ins}(v_t) = T \qquad (5.4)$$

As an example, based on our evaluation data (see Sect. 7.3), we tested different $T$ and obtained best results for $T = 0.74$. Moreover the similarity of edges is reflected by their label similarity, because our experiments showed that this allows for better results than a numeric cost function.

The parameters $\alpha$ and $\beta$ denote a weighting between the structural and linguistic similarity and can be set constant, e. g. to 0.7 and 0.3. For a dynamic parameter calculation we adopted the *Harmony* approach of Mao et. al [101]. Since the problem is to find the maximum number of 1:1 matches between vertices, Harmony aims at calculating unique matches by evaluating combinations which allow for maximum similarity. This is computed in a matrix selecting the maximums of rows and columns deriving the ratio of unique and total matches. Indeed, we noted an improvement of more than 1 % using Harmony over a fixed configuration considering the corresponding minor thesis [58] and our evaluation in Chap. 7.

**2. (b) Additional processing** The original algorithm only considers edges contained in biconnected components of a graph, because only then an ordering can be defined. That means, only vertices reachable by at least two edges are considered. Consequently, all attributes would be ignored. Therefore, we extended the algorithm to match these vertices after the cyclic string edit distance for a specific vertex combination has been calculated. This only affects the complexity additively so it still is nearly quadratic.

**3. Add new matches** The processing is performed until no more matches can be found, which occurs if no neighbours exist or they do not lead to a match. Thereby, we added a removal of invalid matches, i. e. when a class is matched on an attribute or a package vertex the similarity results in zero. The same penalty applies to the match of opposite inheritance or containment directions. Finally, all unprocessed vertices and edges are removed, thus constructing a common subgraph based on the minimal GED.

Finally, the output matches are created based on the matches calculated before. The ouput is formed by all matches of the edit path calculation and all corresponding similarity values.

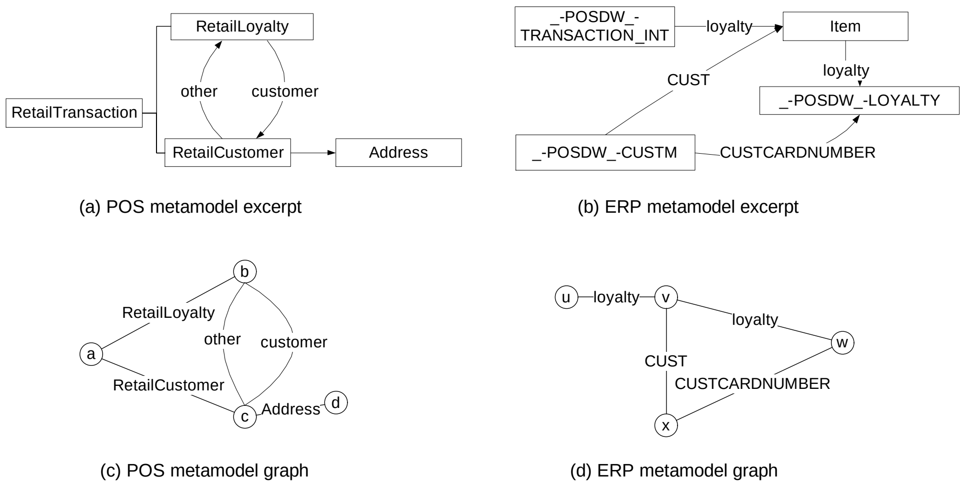


Figure 5.2: Example metamodel excerpts and corresponding graph representations

### 5.1.3 Example calculation

In the following we will present an illustrative example for our planar graph edit distance calculation. The example is taken from the retail store integration scenario as given in Sect. 3.1.1. A reduced version of the POS and ERP metamodel is depicted in Fig. 5.2 (a) and (b). At the bottom the corresponding graph representation of the excerpts can be found. Thereby, the graph (c) corresponds to the excerpt (a) and (d) to (b) respectively. In both excerpts we removed the attributes for a better overview. For simplicity we removed the labels of the classes and replaced them by the letters $a$, $b$, $c$ and $d$. Still the information is preserved by the corresponding edge's label.

The graph representation will be used in the example calculation which is given in Fig. 5.3. The example depicts three steps of the edit distance calculation with the similarity values of the elements calculated. These similarity values represent the costs of relabeling an edge and thus determine if a path should be followed. The root package vertex and thus starting point is given with the elements $a$ and $u$, which are highlighted by a bold border in Fig. 5.3 at the top.

**(i) First iteration** The first step of the algorithm is to examine the neighbourhood of $a$ and $u$ by following all outgoing edges. In the Figure this is highlighted by dark grey. The labels of the edges are given and represent edge and reference names or, if none are present, the target element's name. In the example both containments from $a$ to $b$ and $a$ to $c$ have no name, therefore the edges are labelled according to the target element's labels Re-

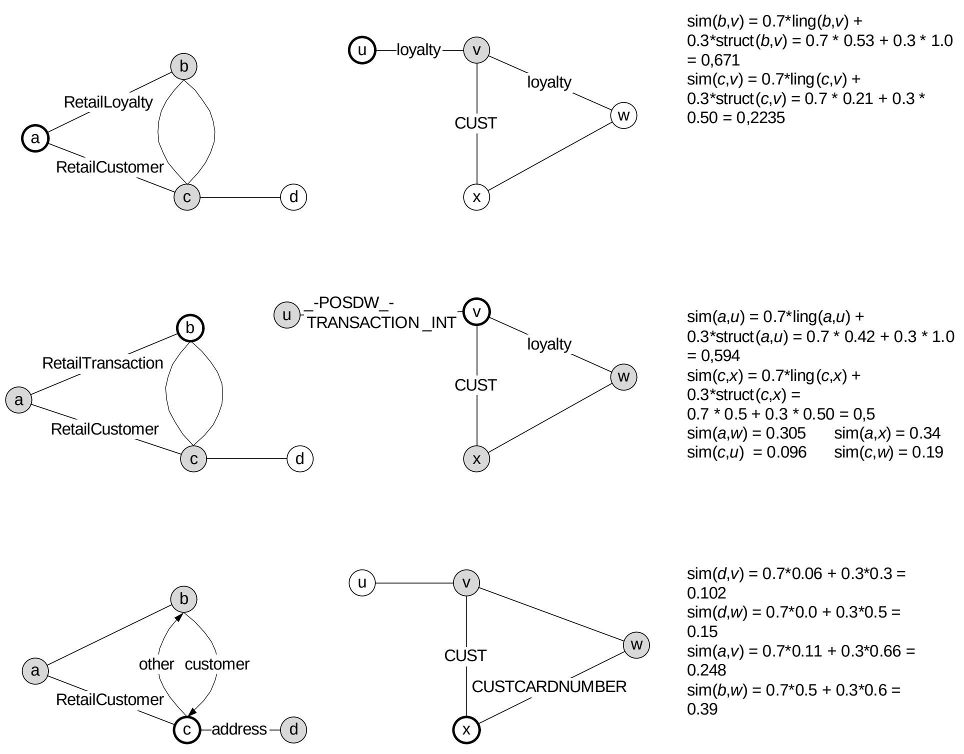

a
b
c
d
RetailLoyalty
RetailCustomer
u
v
w
x
loyalty
loyalty
CUST
sim(b,v) = 0.7*ling(b,v) + 0.3*struct(b,v) = 0.7 * 0.53 + 0.3 * 1.0 = 0,671
sim(c,v) = 0.7*ling(c,v) + 0.3*struct(c,v) = 0.7 * 0.21 + 0.3 * 0.50 = 0,2235
RetailTransaction
RetailCustomer
_-POSDW_-TRANSACTION_INT
loyalty
CUST
sim(a,u) = 0.7*ling(a,u) + 0.3*struct(a,u) = 0.7 * 0.42 + 0.3 * 1.0 = 0,594
sim(c,x) = 0.7*ling(c,x) + 0.3*struct(c,x) = 0.7 * 0.5 + 0.3 * 0.50 = 0,5
sim(a,w) = 0.305
sim(a,x) = 0.34
sim(c,u) = 0.096
sim(c,w) = 0.19
other
customer
RetailCustomer
address
CUST
CUSTCARDNUMBER
sim(d,v) = 0.7*0.06 + 0.3*0.3 = 0.102
sim(d,w) = 0.7*0.0 + 0.3*0.5 = 0.15
sim(a,v) = 0.7*0.11 + 0.3*0.66 = 0.248
sim(b,w) = 0.7*0.5 + 0.3*0.6 = 0.39

Figure 5.3: Example calculation of planar graph edit distance

tailLoyalty and RetailCustomer. In contrast, the edge connecting *u* and *v* is already labelled as loyalty.

After identifying the vertices' neighbours the similarity of these source and target elements is calculated, i. e. *c* and *v*, and *b* and *v* have to be compared. The similarity between *b* and *v* is determined by the weighted sum of the linguistic and structural similarity. The linguistic similarity of RetailLoyalty and loyalty is 0.53 using the Levenshtein distance [153] as also used in our name matcher. The structural similarity is determined by the ratio of three edges of *b* and three edges of *v*, which calculates as 1. For our example we use the following parameters: $\alpha = 0.7, \beta = 0.3$ and $T = 0.5$. As a consequence the weighted sum is calculated as $0.7 \cdot 0.53 + 0.3 \cdot 1 = 0.67$. Since $0.67 > 0.5$, the pair (*b*, *v*) is put into the list of matches and will be considered in the next iteration. The pair (*c*, *v*) yields a similarity of 0.22 and will be discarded.

**(ii) Second iteration** The next iteration starts with the previously obtained pair (*b*, *v*) again highlighted by a bold outline. The pairs for comparison are all combinations of *a*, *c* and *u*, *w*, *x*. Again these pairs are added to the list of matches and are part of the algorithm's output.

**(iii) Third iteration** Finally, we consider the following iteration of (*c*, *x*) where already calculated pairings are considered and combined with the new calculated pairings of (*a*, *v*), (*b*, *w*), (*d*, *v*) and (*d*, *w*) for the final output.

In summary, the algorithm first calculated a correspondence between (*b*, *v*) and extended it to the pairs (*a*, *u*), (*c*, *x*). Along with these pairings the similarity values form the output of the planar graph edit distance. This calculation does not retrieve all mappings, because it misses the pairing (*b*, *w*). Therefore, we investigated further improvements of the algorithm as given in the subsequent section.

### 5.1.4 Improvement by k-max degree partial seed matches

Neuhaus and Bunke stated possible improvements by one seed match [113], because the algorithm relies on its starting point, so we had a closer look at how to handle one or more seed matches. Following this, we investigated two possible approaches: (1) with or (2) without user interaction. The approach without user interaction is to use the default root package matches (one seed match). However, taking user interaction into account allows for further improvement in quality, but requires an additional effort, because a user has to define mappings for a given source and target mapping problem. Therefore, we decided to follow a simple and scalable approach which grants a maximum of information provided.

To keep the interaction simple, we require the user to only give feedback once on a reduced matching problem. Consequently, we present the user with a selection of source and target elements rather than letting him decide on the whole range of elements. That means, the user only needs to match a subset. The subset calculation by $k$-max degree follows a scalable approach by applying a simple heuristic. $K$-max degree is based on the rationale that the seed mapping elements should have the most structural information. Investigating our evaluation data we noted a normal distribution of edges per node, so the probability for vertices to match is highest if they have the highest degree and thus structural information. The computation can be done in linear time and therefore provides a scalable approach.

**Matchable elements selection** We propose to select the $k$ vertices with a maximal degree, i. e. the ranking of $k$ vertices which contain most structural information. So for a given $k$ and graph $G(V,E)$ we define $k$-max degree as:

$$degree_{max}(G,k) = \{v_1, v_2, \ldots, v_k \mid degree(v_i) \geq degree(v_{i-1}) \wedge v_i \in V\} \tag{5.5}$$

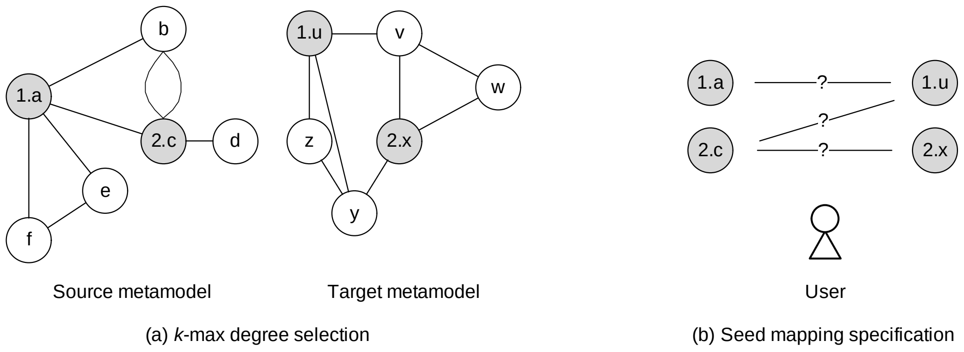


Figure 5.4: Illustrative example of $k$-max degree selection and user interaction by mapping specification

**Seed matches** The user defined mappings yield information on matches and no-matches which are used for the planar GED matcher. These seed matches are included into the algorithm by adding them to the list of all matches. The planar GED matcher makes use of them during the graph edit path calculation as well as during the cyclic edit distance calculation. That means the calculation of a match as in Alg. 5.1 line 5 returns the value 1.0 for the seed matches or 0 for non-matches.

An example of the $k$-max degree seed match selection and specification is depicted in Fig. 5.4, the graphs are abstract representations of the former example of the retail store integration and edit distance calculation. In the example we choose $k = 2$, i. e. we select the two vertices with a maximum degree. The first Step (a) is to select the relevant vertices in the source and target metamodel. The degrees of the vertices *1.a* and *2.c* are both 4, therefore they are selected as mapping candidates. In the target metamodel the selection results in the vertices *1.u* and *2.x* both have a degree of 3.

The selection is followed by the mapping specification of a user as given in Step (b) in Fig. 5.4. The nodes with the maximum degree are presented as candidates and a user has to specify mappings between them. These mappings as well as the information regarding non-matches, i. e. matches which were dismissed by a user are used for the graph edit distance calculation. The results show a considerable improvement which will be discussed in our evaluation (Sect. 7.5.1.4).

## 5.2 Graph Mining Matcher

The GED calculates a global graph-based similarity but does not consider redundant information. To actually exploit redundant information, i. e. patterns, it first has to be discovered in a metamodel and second it has to be mapped. We call the discovery phase pattern mining and split the mapping

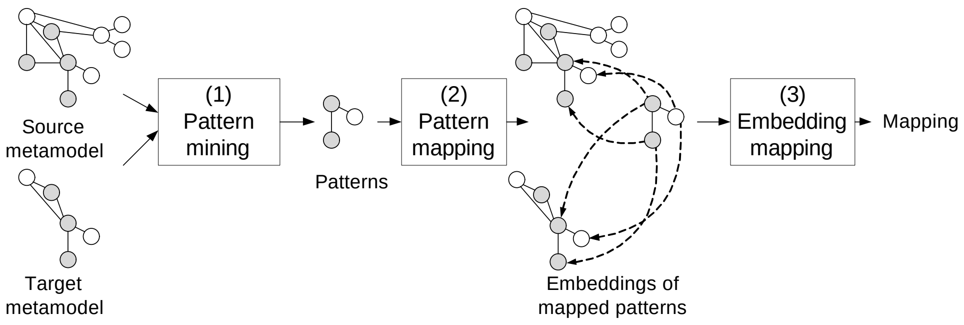


Figure 5.5: General mining based matching process

phase into pattern mapping and embedding mapping. Consequently, we propose three phases of mining based matching, namely:

1. Pattern mining,
2. Pattern mapping,
3. and embedding mapping.

These three phases are depicted in Fig. 5.5. Since graph theory provides mature algorithms for pattern mining we propose to apply generic graph mining algorithms in the first phase. In this phase a set of reoccurring patterns and their embeddings for two given input metamodels are identified. In the second phase we propose to map the patterns of the set onto each other, identifying possible correspondences. Thereby, several matching techniques can be used. The third step of embedding mappings calculates the actual output mapping. This is done by comparing the embeddings of mapped patterns with each other in order to discover element mappings.

Reoccurring information can occur for two reasons, design patterns or redundant modelling. Therefore, we propose two mining approaches for these two kinds of patterns. We base our design pattern (DP) matcher on gSpan [157] and our redundancy matcher on GREW [88]. We adopted and integrated both algorithms in our matching process as described before and extended them to handle metamodel graphs and especially to take linguistic information into account.

Thereby, the algorithms operate on a common graph model, which is a directed labelled (typed) graph. The graph model defined in Sect. 2.2.2 is extended to reflect the labels as type classes. The type classes are used to represent linguistic information, since the algorithms rely on precomputed similarity classes (edge and vertex types) while mining. In the following we present our analysis of graph mining algorithms and justify our choice of gSpan as well as of GREW.

### 5.2.1 Analysis of graph mining algorithms

Following our structure of the GED we first provide our requirements on mining algorithms for the purpose of metamodel matching. We then provide an overview on graph mining algorithms separating them according to their setting of either single or graph transaction setting. This is followed by a comparison of the algorithms presented to derive the base algorithms for our design pattern and redundancy matcher.

#### 5.2.1.1 Requirements for metamodel matching

Patterns in metamodels occur for two reasons, either they are the result of applying design patterns [43] or redundant modelling. Thus, the first requirement (R1.1), as formulated in our problem analysis in Sect. 3.2.3, is to exploit exactly that information. Since the algorithms should lead to an improvement (R1.2) of quality they should make maximal use of the available information and thus support metamodel graph properties that are directed labelled graphs including reflexive edges (self references). An additional consequence is the requirement to save embeddings during pattern discovery because the final output mapping is calculated between elements, i. e. embeddings.

The third requirement (R1.3) of Sect. 3.2.3 is a maximum quadratic complexity in runtime. This requirement cannot be fulfilled by graph mining algorithms, because they inherit NP-complete complexity from subgraph isomorphism detection[14][1]. The reason for this is given by the task of discovering frequent patterns in one or more graphs. Thereby, a starting pattern has to be generated, incrementally extended, and tested for subgraph isomorphism on input graphs. The consequence is an overall exponential runtime due to the isomorphism tests of candidates generated [14]. Moreover, the complexity of mining algorithms depends on the input graph size, frequency of patterns, pattern size, etc. This makes a complexity analysis a difficult task which has not been done so far.

However, there has been work in comparing mining algorithms for common data sets as in [103] that shows results for both runtime and memory. Consequently, we state the requirements of low runtime and memory consumption for an algorithm relative to the known approaches. To summarise, our requirements for graph mining algorithms are:

1. Exploit redundant and design information,
2. Saving of pattern embeddings,

[1]There is one mining algorithm that makes use of planarity. However, it has been shown that such an approach suffers from a high constant $c$ for $O(c \cdot n^2)$ and is only feasible for pattern graphs with more than 800 nodes [86]. The patterns mined are much smaller and thus the approach is not applicable.

3. Support of metamodel graph properties,
4. Low runtime,
5. Low memory consumption.

#### 5.2.1.2 Overview on graph mining algorithms

An overview and comparison of graph mining algorithms is depicted in Tab. 5.2. The first group on top of the table depicts an overview on the popular single graph setting algorithms. For a complete survey please refer to [89]. As confirmed by [14, 89] the most popular and first single graph setting algorithm is SUBDUE [1]. The original algorithm was presented in 1994 and has been improved since [79]. The algorithm follows a greedy search by incremental pattern extension. Thereby, the extended patterns are checked for their compression of the input graph. That is the replacement of all pattern embeddings by one vertex and the final graph size. The best compressing patterns form the final output.

The GBI [106] algorithm follows a similar principle of replacing structures (reduction) but does not check for compression but rather for the entropy of the patterns.

GREW [88] is dedicated to pattern extraction of one large graph. So far complete approaches like SiGram [89] are not applicable to large graphs (with more than 100 vertices). GREW is approximate so it is applicable in large-scale scenarios while still remaining memory sufficient.

The graph transaction setting deals with pattern mining on a set of graphs. Several algorithms have been proposed as given by a survey in [14], therefore we only present a selection based on popularity and acceptance in the second group in Tab. 5.2.

The first graph transaction setting mining system was proposed in AGM [66], which introduced an apriori level-wise approach. Thereby, all patterns which are one vertex serve as starting points. These patterns are merged recursively where each step forms a level in the pattern search. Each merging is followed by a check for embeddings, i. e. the pattern is validated. The maximal valid patterns form the final output. To prevent redundant checks patterns are encoded and merged according to certain rules. This approach has been further refined by FSG [87] by optimizing the performance, as was done in FFSM [65]. gSpan [157] followed a similar approach but introduced a depth first search based encoding system of the graphs and patterns, which is more efficient in terms of computation time and memory [79]. A further improvement of gSpan can be found in CloseGraph [158] which compresses patterns and thus reduces the memory consumption.

In [103] these mining systems have been compared on a common data set and implementation, concluding that gSpan is the fastest and most memory efficient system.

| Algorithm | Nature | Memory | Runtime | MM Graph | Emb. |
|---|---|---|---|---|---|
| **Single graph** | | | | | |
| SUBDUE [1] | approx. | + | – | ✓ | ✓ |
| GBI [106] | approx. | – | – | ✓ | ✓ |
| SiGram [89] | complete | – | + | ✗ | ✓ |
| *GREW* [88] | approx. | + | ++ | ✓ | ✓ |
| **Graph transaction** | | | | | |
| AGM [66] | complete | + | – – | ✓ | ✗ |
| FSG [87] | complete | + | – | ✓ | ✓ |
| FFSM [65] | complete | – – | + | ✓ | ✓ |
| gSpan [157] | complete | + | + | ✓ | ✓ |
| *CloseGraph* [158] | complete | ++ | + | ✓ | ✓ |

Table 5.2: Overview and comparison of graph mining algorithms; ✓/✗– support/no support; -/+ – more/less memory/runtime

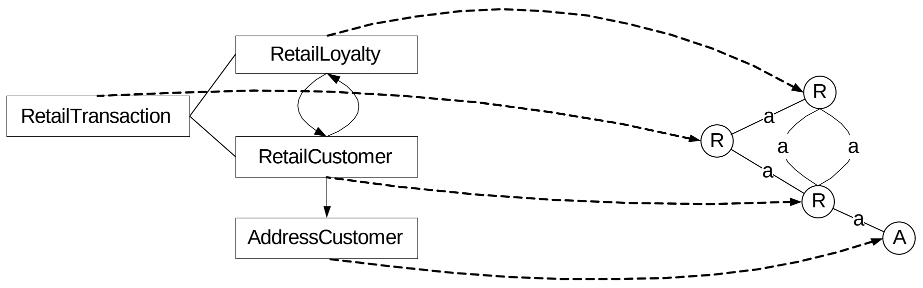
### 5.2.1.3 Summary

We selected two algorithms to serve as a basis for mining based metamodel matching. We chose two algorithms because of the two reasons for patterns in metamodels that are (1) redundant information and (2) design patterns.

(1) Redundant information is redundant if it occurs more than once in one metamodel, thus we investigated algorithms of the single graph setting. Since graph mining algorithms show an exponential complexity we weighted runtime and memory as most important. GREW shows the best runtime compared to the other algorithms [88] and is especially applicable to large-scale scenarios in contrast to SiGram, which due to its complete nature needs to save every embedding, which results in an increased memory consumption. Since the other algorithms [79, 106] are also approximate and show a worse runtime than GREW and they are not scalable [88, 102], GREW consequently forms the basis of our redundancy matcher.

Since (2) design patterns occur across metamodels we selected an algorithm from the graph transaction scenario. Again memory consumption and runtime are of interest as well as support for metamodel graphs and embedding saving. AGM [66] is not capable of saving embeddings and needs to be discarded. Furthermore, a study [103] compared the gSpan, FSG, and FSSM algorithms concluding that gSpan outperforms the others w.r.t. memory and runtime. The proposed enhancement of gSpan, i. e. CloseGraph [158], even improves the memory consumption while preserving its runtime. Therefore, we chose the algorithm as a basis for our design pattern matcher.

In the following we will present the graph model on which both mining matchers operate.



$P_1$ = (RetailTransaction , RetailLoyalty ) = 0.47
$P_2$ = (RetailTransaction , RetailCustomer ) = 0.41
$P_3$ = (RetailTransaction , AddressCustomer ) = 0.12
$P_4$ = (RetailCustomer , AddressCustomer ) = 0.57
t=0.4 $S_1$= R = ($P_1$, $P_2$), $S_2$ = A = ($P_4$)

Figure 5.6: Example for mining graph model types using similarity classes

### 5.2.2 Graph model for mining based matching

The mining of metamodels is done using a typed graph model. Thereby, all vertices and edges are grouped into classes of similar elements to encode linguistic and metamodel type information. The final type of an element $e$ is a tupel $(t, S)$ where $t$ is the metamodel type of the element, such as class or attribute, and $S$ a similarity class.

A similarity class $S$ is defined by a two step process. First, all elements are sorted according to their type, i. e. class, reference, operation, attribute or package. These groups of different element types are then split into subsimilarity classes based on name similarity. Thereby, a pair of elements $(e_i, e_j)$ is assigned to a similarity class if $sim(e_i, e_j) > t$ with $t$ being a fixed threshold[2] and $e_i$ the first element in a type class $t$. The similarity class assignment is repeated for all assigned elements $e_j$ and unassigned elements $e_u$, this time multiplying a potential error by the following condition $sim(e_i, e_j) \cdot sim(e_j, e_u) > t$. The multiplication allows to prevent too large similarity classes.

For instance, let $sim(a, b) = 0.7, sim(b, c) = 0.7$ and $t = 0.7$, and to demonstrate the intransitivity of string similarity let $sim(a, c) = 0.2$. If we only apply a fixed threshold, $a, b$ and $c$ would be added in two steps to the same class. However, if we take the error of a similarity of 0.7 into account $sim(a, b) \cdot sim(b, c) = 0.49$, $c$ would not be part of the class, but rather create a new one. Finally, all similarity classes are assigned to a label, thus establishing the similarity classes $S$.

Figure 5.6 depicts an example for the similarity class calculation and the resulting graph. On the top left the example metamodel is depicted and on the top right the resulting graph with the similarity classes as labels.

[2]The evaluation in the context of a master thesis [112] showed that $t = 0.7$ yields the best results in terms of correctness (precision) and completeness (recall).

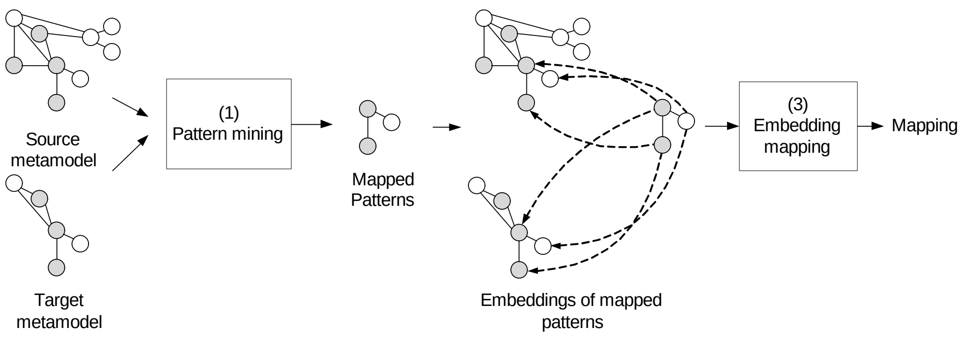


Figure 5.7: Design pattern matcher process

The dashed lines represent the correspondences between elements. On the lower left the similarity class calculation is shown and on the lower right the resulting classes, thereby pairings of elements are noted as $P_i$.

The example threshold has been set to $t = 0.4$, so the similarity classes $S_1 = (P_1, P_2)$ result in the first comparison of all classes with the first element of the type class, that is RetailTransaction. In absence of our error propagation the second iteration would add the class AddressCustomer because of its similarity to RetailCustomer. This is not desirable, because $P_2 \cdot P_4 = 0.23$, therefore $P_4$ is assigned to a new similarity class $S_2 = (P_4)$. $S_1$ is assigned the label $R$ and $S_2$ the label $A$ which are used to type the vertices. For simplicity we omitted attributes and references, but they are considered in the same manner.

The resulting type graph is labelled according to the metamodel types ($t$) and the similarity classes ($S$) are used by the design pattern matcher and the redundancy matcher for mining patterns. The linguistic similarity is ensured by the previous similarity class calculations. In the subsequent sections we will describe our two mining matchers in detail along the three steps of mining based matching mentioned before.

### 5.2.3 Design pattern matcher

One type of patterns are design patterns which occur in both the source and the target metamodel. Therefore, we propose an approach which searches for patterns in both metamodels simultaneously. We base our approach on gSpan [157], which has the main idea of mining by incremental pattern extension. Starting with a trivial pattern of one edge, it is extended incrementally until no further extension is possible. The possibility of extensions is defined by the occurrences (embeddings) in both metamodels.

The design pattern matcher process is depicted in Fig. 5.7. First, both graphs are searched for common pattern, thus the pattern mappings step is obsolete, because a pattern is only found if for both graphs an embedding exists. The subsequent mapping of the patterns embeddings yields the final mappings. The algorithm as given in Alg. 5.2 performs the pattern identification and mapping and finally the mapping of the pattern embeddings.

The first step is a pre-processing (Alg. 5.2, line 3). Two input metamodel graphs are processed as described in the previous section for establishing similarity classes, which are the types. These types allow for isomorphism tests by structural and linguistic information. Next, all frequent edge types (edges having a frequency of more than one) are determined and used for the actual pattern mining (line 4).

**Algorithm 5.2** Design pattern matcher algorithm

```
 1  input: source $G_s(V_s, E_s)$, target $G_t(V_t, E_t)$
 2  Variables: pattern set $found$
 3  pre-process $G_s$, $G_t$
 4  for each $e_f \in \{e | freq(e) > 1 \wedge e \in E_s, E_t\}$
 5    Add patterns of minePatterns($e_f$) to $found$
 6    Mark edge as visited
 7  for each $p \in found$
 8    if p is relevant
 9      Add $(emb(p, G_s), emb(p, G_t))$ to output mapping
10  return output mapping
```

### 5.2.3.1 Pattern mining

The pattern mining follows steps proposed by gSpan [157] which are shown in pseudo code in Alg. 5.3. In detail these steps are:

1. Mine for design patterns
   (a) Mining of patterns by incremental extension of frequent edges
   (b) Repeat pattern extensions until no frequent pattern is found
2. Filter patterns

The first pattern identification step is *(1) the pattern mining*. Given an ordered list of frequent edge types, the most frequent edge type is selected as the first pattern. Starting with this pattern of an edge a possible extension on the basis of all embeddings in the first graph ($G_s$) is calculated (Alg. 5.3, line 3). Second, it is checked if this extension is also possible in the second graph $G_t$. That means, if there exist embeddings of this pattern in $G_t$ (line 5) the pattern will be extended if possible, i. e. an edge will be added to the pattern (and checked in all its embeddings) (Alg. 5.3 line 6). The extension steps are repeated for all frequent edge types until all have been processed.

**Algorithm 5.3** Pattern mining in design pattern matcher (minePatterns)

```
input: $G_s$, $G_t$, pattern $p$
output: pattern $found$
$ext \leftarrow$ extensions of $p$ in $G_s$
for each $x \in ext$
  if $x$ exists in $G_t$
    x.embeddings $\leftarrow$ $emb(x, G_s) \cap emb(x, G_t)$
if $x$ is closed
  add $x$ to $found$
for each $p_x \in ext$
  if $p_x$ is extendable
    minePatterns($G_s, G_t, p_x$)
return $found$
```

Such an *(1.a) extension of a pattern* is built by adding an edge under the restriction of a depth first search (DFS) tree. That means, each edge to be added is enumerated, and the resulting graph has to comply with an enumeration imposed by a DFS tree. Additionally, back-edges have to be inserted first, i. e. edges that are not part of the DFS-tree but rather point from a DFS tree node back to another one. This DFS restriction as given in the original algorithm allows to define an order on the extensions possible and thus reduces the search space [157]. If an extension only occurs in one graph the pattern is not frequent and all possible resulting extensions can be discarded. This is due to the antimonotonicity of the frequency measure, which states that an infrequent subgraph cannot become frequent by adding further edges [157].

The *(1.b) extension of the current pattern is repeated* until no embedding can be found in the source or target graph. Thereby, each extension pattern and the corresponding embeddings are saved. To reduce memory consumption we apply a compression technique, the so-called closed graphs [158]. That means, if an extension of a pattern occurs in every embedding, the unextended pattern does not need to be saved along with its embeddings, because it is a subset of this extension. This condition is called equivalent occurrence, meaning that the count of all embeddings of a pattern and the count of all embeddings of the pattern extended by an edge is equal. Applying this technique allows for reduced memory, thus making the approach applicable for larger metamodels (more than 1,000 elements).

In the following iteration the next frequent edge type is chosen and again extended as given in Step 1 a.

The resulting patterns can contain trivial and misleading patterns, leading to misleading matches which may affect the result quality. In addition, the exponential runtime complexity depends on the number of patterns as well as their size.

For an improved result quality and runtime complexity we propose to *(2) filter the set of found patterns* based on their relevance (Alg. 5.3 line 8). For the relevance calculation we propose a weighted approach which relates the size and frequency of a pattern. The relevance of a pattern $p$ is calculated by

$$r(p) = |p|^{\alpha} \cdot freq(p)^{\beta}$$

where $|p|$ is the size of a pattern (number of edges and vertices) and $freq$ its frequency (number of embeddings). The parameters $\alpha$ and $\beta$ determine a weight for the influence of size or frequency of a pattern. For instance, if $\beta$ is negative, the frequency has to be small for higher relevance.

#### 5.2.3.2 Pattern mapping

Since patterns are only mined as valid patterns if they occur in both graphs, i. e. there exist embeddings, there are no patterns exclusive for a source or target metamodel. As a result there is no need to map patterns but only to map the respective embeddings.

#### 5.2.3.3 Embedding mapping

The previous mining step calculated patterns and their corresponding embeddings in the source and target graph. Consequently, each combination of source and target embeddings of the same pattern map onto each other. Therefore, we propose to create the cartesian product of the source and target embeddings as output mappings (Alg. 5.3 line 9–10). These mappings are on pattern level and thus between subgraphs. The element-wise mappings do not need to be calculated because they are defined by their respective position in the pattern they belong to.

#### 5.2.3.4 Example

We will work on the example depicted in Fig. 5.8 to demonstrate the principle of our design pattern matcher. Imagine two metamodels in a typed graph representation as described in Sect. 5.2. There exist the following similarity classes for the source and target metamodel: $A, R$ for classes and a solid or dashed line for references. For the sake of simplicity we depicted reference types as lines and skipped attributes. The similarity classes (types) are used to determine valid assignments and thus extensions, i. e. only elements of the same type are assignable.

**(a) Start pattern** The pattern mining step processes all edge types, so we depict one exemplary type which is the solid line. The pattern of two vertices and an edge of type solid line is shown on the top left (Fig. 5.8 a).

The vertices are enumerated according to a DFS convention. The embeddings of this pattern are highlighted in the source and target metamodel by bold lines, thus the pattern has one embedding in each metamodel. Please note that we only highlighted one embedding in the example for a better overview, indeed there exist three embeddings for the source metamodel and two for the target metamodel.

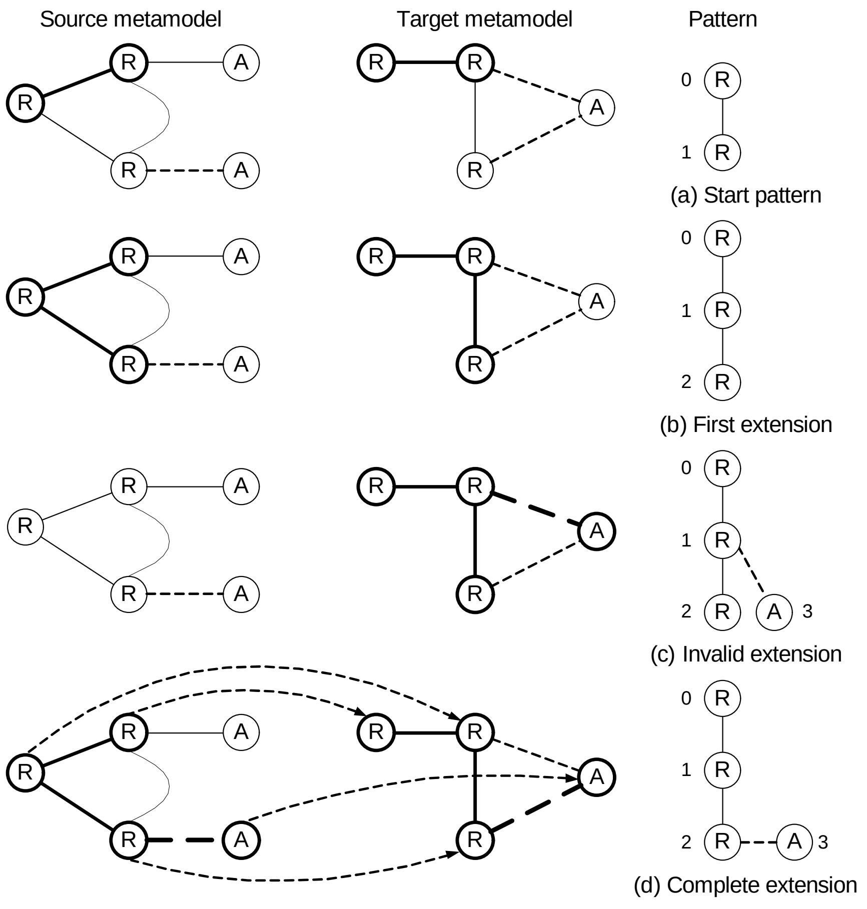


Figure 5.8: Example for pattern mining by the design pattern matcher

**(b) First extension** The first extension of the pattern by an edge is shown in Fig. 5.10 (b). This extension is valid, because in each metamodel exists one embedding of the extended pattern. Again the nodes are labelled according to a DFS convention.

**(c) Invalid extension** The next extension shows the negative case with an invalid extension (Fig. 5.8 c). The pattern is extended according to the DFS convention by a fourth vertex with the dashed line type. This pattern has an embedding in the target graph as highlighted in bold. However, there is no embedding in the source graph for the given edge type. This particular extension and all resulting ones are discarded for further mining.

**(d) Complete extension** An example for a complete extended pattern is shown at the bottom of Fig. 5.8. The pattern has been extended by a fourth node with the dashed line type which has embeddings in both metamodels. Finally, the pattern already determines the mappings between the source and target elements by the position of the vertices within the pattern. We depicted the mapping between those elements by a dashed line with an arrow. These mappings form the output of the pattern mapping step and are used for similarity value calculation for all elements, e. g. the name path matcher (see Sect. 7.2).

#### 5.2.3.5 Remarks

The graph mining algorithm has to solve two fundamental problems. First, it has to ensure DFS-tree conforming extensions. This is done by a canonical code building, which is an NP-complete problem and exponential in the size of the patterns. Second, all embeddings of a pattern, i. e. all subgraph isomorphisms, have to be calculated, which is also NP-complete.

The algorithm's complexity can be bounded by $O(k \cdot f + r \cdot f)$. Thereby, $f$ is the count of frequent subgraphs, $k$ is the maximal count of subgraph isomorphisms of a pattern and $r$ is the count of non-DFS conforming extensions that have to be filtered. That means, $k \cdot f$ bounds the maximal number of subgraph isomorphism computations while $r \cdot f$ bounds the number of extension filterings.

Consequently, the algorithm needs to be reduced in runtime in order to make it applicable in metamodel matching scenarios. Therefore, we proposed to increase the number of edge and vertex types by similarity class calculation. This decreases $k$ because a pattern has fewer isomorphisms in the presence of more types. We also limited the degree of a vertex to $d$, which leads to a limitation of possible extensions. Finally, we also introduced a maximum pattern size, to reduce the filtering calculations.

### 5.2.4 Redundancy matcher

Since redundant information occurs as frequent subgraphs in one graph we propose to mine for redundant information based on the established approximate graph mining algorithm GREW [88]. The basic idea is to reduce a graph by removing edges of one type and merging their connected vertices. This is based on the edge contraction principle as defined in Def. 9 in Sect. 2.2.3.1. Interestingly, edge contraction is contrary to the design pattern mining, which incrementally extends a pattern. In contrast, edge contraction

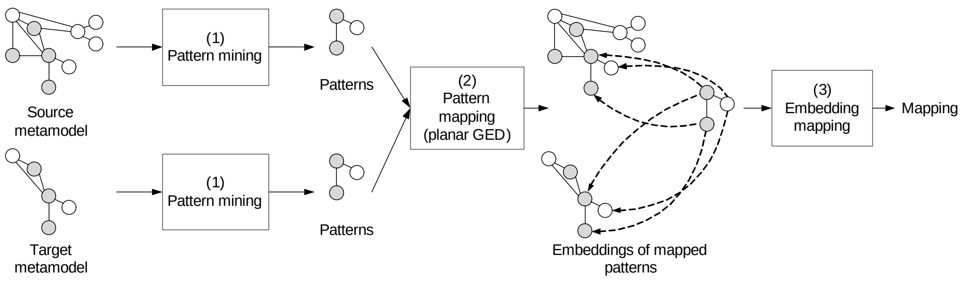


Figure 5.9: Redundancy matcher process

reduces the graph step-wise until no further reduction or contraction is possible, thus attributes are merged into classes and classes with classes etc.

Figure 5.9 depicts the matching process of the redundancy matcher. In contrast to the design pattern matcher the mining is not performed on the two metamodel graphs simultaneously but independent from each other. The patterns extracted are compared to each other in a second step using our planar graph edit distance and finally their embeddings are mapped.

Algorithm 5.4 shows the redundancy matcher's steps. The pre-processing (line 3) is the same as for the design pattern matcher and creates a typed graph using the similarity class calculation as described in Sect. 5.2. Lines 4 and 5 show the independent pattern identification, whereas line 7 specifies the distance computation between all patterns identified. Finally, the mappings are created as given in line 8 and 9 respectively. In the following sections we will detail each of these steps.

**Algorithm 5.4** Redundancy matcher algorithm

```
1   input: source $G_s(V_s, E_s)$, target $G_t(V_t, E_t)$
2   variables: source patterns $found_s$, target patterns $found_t$
3   pre-process $G_s, G_t$
4   add relevant patterns in minePatterns($G_s$) to $found_s$
5   add relevant patterns in minePatterns($G_t$) to $found_t$
6   for each $p_s$ in $found_s$
7     for each $p_t$ in $found_t$ with minimal distance to $p_s$
8         create mapping ($p_s$.embeddings, $p_t$.embeddings)
9   return output mapping
```

### 5.2.4.1 Pattern mining

The pattern mining as given in the adapted algorithm GREW [88] and shown in Alg. 5.5 comprises the following steps:

1. Mine for redundant patterns

   (a) Determine the most frequent edge type for contraction
   (b) Contract independent edges if frequent
   (c) Repeat until no further contraction is possible

2. Repeat mining for redundant patterns with new edge type

3. Filter patterns

The input graph of the source metamodel is mined for redundant information by *(1 (a)) determining the most frequent edge type* as start type for frequent types. The type $t_e$ of an edge $e = (v, u)$ connecting the elements $v$ and $u$ is defined as $t_e = (e, t_u, t_v)$, i. e. by the edge itself and the incident vertices. An edge type $t_e$ is frequent if $|emb(t_e)| > f_{min}$, i. e. if more than $f_{min}$ non-overlapping edges of type $t_e$ exist. The starting type is the one with the maximal occurrences (Alg. 5.5, line 7). For this frequent edge type $t_f$ an overlaying graph $G_o$ is constructed. A vertex in $G_o$ represents an occurrence of the edge type. An edge in $G_o$ is created, if two occurrences share a vertex in $G$.

If two edges to be contracted are adjacent one needs to be chosen. For best result quality we propose to apply a maximal independent set calculation of vertices using a greedy algorithm as given in [52] (line 8 and 9), because this aims at a maximal number of contractions and thus maximal pattern size.

**Algorithm 5.5** Pattern identification for redundancy matcher (minePatterns)

```
1  input: Graph $G$
2  variables: $found$
3  $\zeta \leftarrow$ G
4  while frequent edges exist
5    for each $e_f \in \{e|freq(e) > 1$ in $\zeta$ ordered by frequency
6      calculate maximal independent set $M$ for type $t_f$ of $e_f$
7      if $|M| > minFreq$
8        add pattern represented by $t_f$ to $found$
9        mark every edge in $M$
10     contract marked edges in $\zeta$
11 remove marked edges from $G$
12 return found
```

Afterwards, the *(1.(b)) maximal independent set of edges is contracted*. That means the determined edges as well as their incident vertices are replaced by a multi-vertex (line 10). A multi-vertex is a vertex $w$ that represents an embedding of a pattern, i. e. the edge $e$ and its incident vertices $u, v$. The edges incident to $u, v$ will be replaced by multi-edges which are connected to the multi-vertex $w$. Consequently, the multi-vertex is connected with the original graph and represents the original edge and its position in the subgraphs of the incident multi-vertices. By this strategy, smaller patterns are joined to larger patterns in every iteration.

The *(1.(c)) edge contraction is repeated* until no further contraction is possible. This occurs if the graph is reduced to two remaining classes or no independent set can be calculated.

The *(2.) mining process is repeated*, because the results of the algorithm depend on the type chosen for contraction. Consequently, patterns to be found are possibly missed. This behaviour justifies the approximate nature of the algorithm as also noted by the GREW authors in [88]. However, the result quality can be improved by applying the algorithm multiple times on a graph. Thereby, the most frequent non-processed edge type is chosen and the contraction is applied again. This results in different patterns for each iteration. To reduce the overhead for each run, already contracted and thus used edges will be removed from the graph to improve the coverage of the algorithm.

The *(3.) extracted patterns will be filtered* in the same manner as for the design pattern matcher, i. e. based on their relevance.

#### 5.2.4.2 Pattern mapping

Since the patterns have been mined separately for a source and target graph a mapping has to be calculated. Therefore, we propose to apply our GED as described in Sect. 5.1. It is used to calculate a similarity value as well as a mapping between two patterns. This planar graph edit distance determines the distance of two patterns; where the most similar patterns will finally be mapped.

#### 5.2.4.3 Embedding mapping

The embeddings are mapped in the same way as for the design pattern matcher with the additional information of mapped patterns. That means only embeddings of two mapped patterns are matched with each other leading to the final element mappings.

#### 5.2.4.4 Example

We present in Fig. 5.10 an illustrative example for our redundancy matcher. The example shows the complete process of pattern mining, pattern mapping, and embedding mapping. For simplicity, we depicted only one metamodel for mining, which contains a redundant address ($A$). The metamodel has been pre-processed by assigning types to edges and vertices. The vertex type is depicted as a label which is denoted as $R, A$. Edge types are represented by a dashed or solid line.

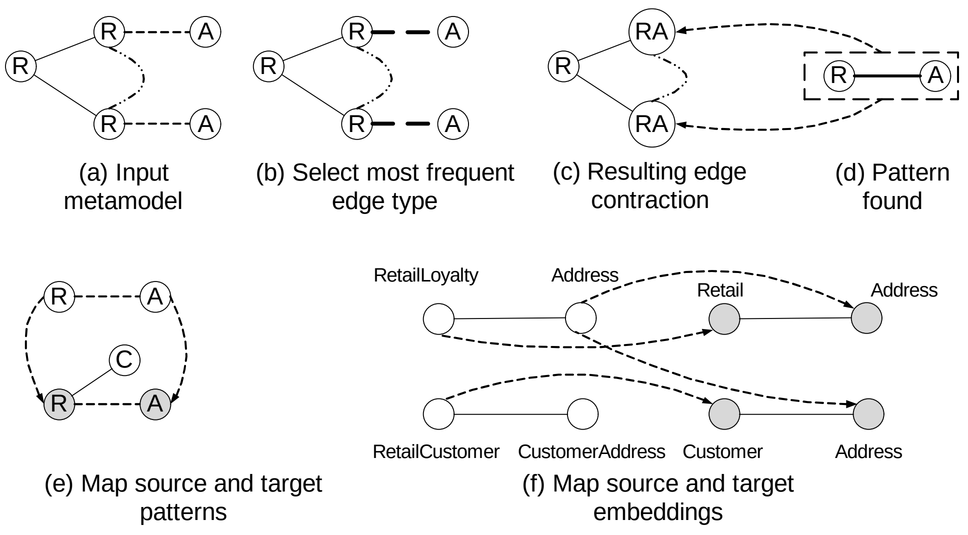

R
R
A
R
A
(a) Input metamodel
(b) Select most frequent edge type
RA
RA
(c) Resulting edge contraction
(d) Pattern found
C
(e) Map source and target patterns
RetailLoyalty
Address
Retail
Address
RetailCustomer
CustomerAddress
Customer
Address
(f) Map source and target embeddings

Figure 5.10: Example for pattern mining by the redundancy matcher

**Select most frequent edge type** The pattern mining begins by selecting the most frequent edge type (Fig. 5.10 a), defined as the type of an edge and of adjacent vertices, which is a dashed line between the vertices of type $R$ and $A$. This type is preferred over the one given by two vertices of $R$ connected by a solid line, because the size of its maximal independent set is less, since the embeddings share one vertex ($R$).

**Contract selected edge type** In the subsequent step all embeddings of this edge type are contracted (Fig. 5.10 b). That means the edges are removed and the adjacent vertices are merged into a new multi-vertex. In our example the dashed line edges are removed and the vertices $R$ and $A$ are merged in the new multi-vertex of the new type $RA$. Since no further contraction is possible this is the resulting pattern as given in (d).

**Map patterns** This pattern has been mined for one metamodel; imagine now an additional pattern from another metamodel, as given in Fig. 5.10 (e). The pattern from the previous steps can be found on the top where on the bottom the example pattern is depicted. Both patterns are mapped using our planar graph edit distance. The resulting mapping is shown as dashed lines between the corresponding elements.

**Map embeddings** Finally, the pattern embeddings have to be mapped on to each other. This step is given in Fig. 5.10 (f). Using an arbitrary matching algorithm the elements are mapped, again depicted by a dashed line. Exactly the mappings between the redundant address elements are the output of the redundancy matcher.

#### 5.2.4.5 Remarks

Again the complexity introduces runtime problems for this matcher. The isomorphism test of patterns is tackled by applying our planar graph edit distance. However, the problem of determining the maximal independent set remains. We tackle that by an approximate greedy algorithm [52]. The last problem of identifying the edge positions is reduced by limiting the pattern size.

## 5.3 Summary

In order to tackle the problem of insufficient matching quality we proposed three approaches which make use of either structural or redundant information: a planar graph edit distance algorithm as well as two graph mining based approaches, the design pattern matcher and the redundancy matcher. We analyzed and adopted existing graph theory algorithms. Table 5.3 depicts a summary of our contributions:

**Planar graph edit distance matcher** The planar graph edit distance algorithm results from a structured analysis of graph matching algorithms. The algorithms suffer from the problem of complexity, hence we deduced a requirement analysis resulting in a selection of a planar graph edit distance algorithm with quadratic runtime for our purposes. We proposed adaptations and extensions of this algorithm to take linguistic and structural information into account and to calculate mappings for metamodel matching. Additionally, we proposed the $k$-max degree approach for further improvement of the matching quality. Thereby, we make use of $k$ user input mappings ranked by the degree of elements.

**Graph mining matcher** Our graph mining matchers are based on the idea of discovering reoccurring patterns in a metamodel for matching. There are two reasons for such patterns, which are the presence of design patterns or in case of redundant modelling. We investigated the state of the art in graph mining and, based on a requirements analysis, selected two approaches for each class of patterns. We proposed two matchers, the design pattern matcher and the redundancy matcher. Both approaches show three steps, which are: pattern mining, pattern mapping, and embedding mapping.

The design pattern matcher mines for patterns based on an incremental extension of a pattern. Checking for each extension if it is a valid pattern, i. e. if it occurs in both metamodels, design patterns are found. The pattern

| Matcher | Contributions |
|---|---|
| Planar graph edit distance | • Adoption and extension of planar graph edit distance for metamodel matching<br>• Definition of a vertex distance function based on structural and linguistic similarity<br>• Dynamic parameter calculation for weights of a distance function<br>• Consideration of relations' direction by similarity value penalties<br>• Similarity calculations during cyclic string edit distance calculation<br>• Processing of one-neighbour vertices<br>• $k$-max degree approach for improved results |
| Design pattern & Redundancy | • Architecture and process for pattern mining algorithms in metamodel matching<br>• Adopted two algorithms for design pattern (gSpan) and redundancy-based (GREW) mining<br>• Definition of typed graph model including linguistic information encoded in similarity classes (types)<br>• Proposed relevance-based filtering of patterns |

Table 5.3: Contributions of structural graph-based matchers

mapping is part of the identification, because a pattern has to occur in both graphs. The final element mapping is calculated using matchers on the elements of pattern embeddings.

The redundancy matcher proposed by us mines two metamodels independently for patterns. The mining is based on the principle of reducibility, reducing the most frequent edge type until no further reduction is possible. This approach is repeated for different edge types. The resulting patterns for both metamodels are then mapped using our GED. Finally, the embeddings of mapped patterns are matched with each other to calculate the output mapping.

# Chapter 6

# Planar Graph-based Partitioning for Large-scale Matching

In this chapter we discuss our solution for the problem of insufficient support for large-scale metamodel matching, i. e. the problems of memory and runtime while mapping metamodels with thousands of elements. An approach to tackle the memory problem is to split the input metamodels into parts and match them independently. We propose to calculate these parts using a planar partitioning approach based on a heuristic element removal and connected component calculation. The subsequent partition merging is based on our proposal of utilizing the structural measurements of coupling and cohesion.

However, partitioning does not decrease runtime because a comparison of all partition elements still leads to the cartesian product of source and target elements. The number of comparisons can be decreased by selecting only a subset of partition pairs for matching in a so-called partition assignment process. The selection and assignment of partitions for matching can reduce runtime but also decrease result quality. Therefore, we present and compare four partition assignment approaches, which are evaluated and discussed in our evaluation.

The subsequent sections present an overview of the partitioning-based matching followed by our planar graph-based partitioning to conclude with a description and comparison of the assignment approaches applied by us.

## 6.1 Partition-based Matching

Especially in industry current matching systems cannot fulfil the demands of matching large-scale metamodels, i. e. metamodels with more than 1,000

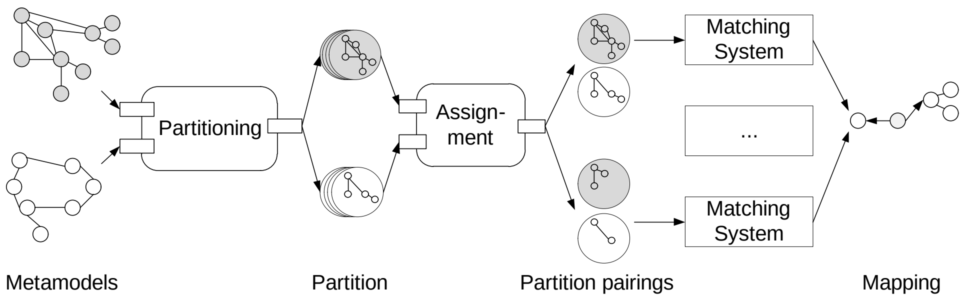


Figure 6.1: Processing steps of partition-based matching and assignment

elements. Either they are not able to match them at all or they require enormous resources for matching. As stated in our problem analysis in Sect. 3.2.1 the main reason for scalability issues of matching systems is the traditional approach of pair-wise element comparison w.r.t. an input source and target metamodel. Calculating element correspondences for all pairs of elements, a system shows at least quadratic growth in memory and runtime.

An approach that is well-known in the area of schema matching [25] is to split the input into smaller parts using clustering and match these parts independently. We adopt this idea for metamodel matching as given in the processing steps overview in Fig. 6.1. There, the input metamodels are split into parts (partitions) in the first step (6.2) by a partitioning component. Partitioning is similar to clustering but offers the advantage of guaranteeing an upper bounded partition size as well as a similar size for partitions. However, partitioning reduces the memory needed but not the runtime. The reason is simple, because matching each element of each partition of the source and the target metamodel with each other, the number of comparisons done is the same as matching pair-wise without partitioning. In addition, the partitioning algorithm itself introduces some computational overhead.

To overcome this problem, promising combinations of source and target partitions are identified and assigned to each other via an assignment algorithm resulting in sets of partition pairs in the second step (6.3). The assignment is done based on a partition similarity, which we propose to calculate using partition representatives instead of performing a pair-wise partition element comparison. The goal is to only match partitions which show a high similarity. To select these pairs we study two similarity threshold-based approaches as well as two algorithms calculating optimal one-to-one or one-to-many assignments. The assigned partition pairs serve as input for an arbitrary matching system and are matched independently.

The first step, the partitioning, is described in the subsequent section 6.2 introducing our planar graph-based partitioning algorithm. Step 2, the assignment, is detailed in subsection 6.3.

## 6.2 Planar Graph-based Partitioning

Inspired by a generic algorithm from graph theory [3] we propose to use an approach that divides an input metamodel into subgraphs, i. e. partitions. Thereby, we exploit structural and metamodel type information. The idea is to first split an input metamodel into partitions until these partitions fall under a pre-defined size limit. The splitting is achieved by removing elements from the metamodel. The remaining elements are ordered according to their connectivity to the partitions obtained. The ordered elements are finally merged with the partitions previously calculated. We propose to merge these partitions and remaining elements based on coupling and cohesion, two structural measurements, to achieve a structure-preserving partitioning.

The partitioning results in a number of output partitions, which do not need to be user-defined since the size of the partitions is already given. In the following sections we will analyse partitioning and clustering algorithms to select the planar graph-based partitioning algorithm.

### 6.2.1 Analysis of graph partitioning algorithms

Partitioning and clustering are similar approaches as described in the background chapter (Sect. 2.2.6). Both calculate the splitting of an input into smaller parts. The main difference is that partitioning aims at parts of similar size, whereas clustering does not.

In the following we will present an analysis of existing graph partitioning and graph clustering algorithms with the purpose of selecting the basis for our partitioning phase. First, we derive requirements in the context of large-scale metamodel matching and then study existing approaches w.r.t. the requirements. Based on this we select the basis for our approach in the subsequent section.

#### 6.2.1.1 Requirements for metamodel matching

The requirements for large-scale metamodel matching state that an algorithm should partition input metamodels (R2.1), reduce runtime (R2.2), and lead to a minimal loss of result quality (R2.3) (see Sect. 3.2.3 ).

The partitioning of an input metamodel should also reduce the number of comparisons performed and thus reduce memory consumption. The memory consumption reduction can be ensured if the input splitting ensures a maximal size per partition. Therefore, the partitioning algorithm has to ensure a maximal size of partitions and consequently a variable number of partitions. To reduce runtime an algorithm should not introduce additional complexity. Since matching has a quadratic complexity a partitioning algorithm may not exceed quadratic complexity. Moreover, a minimal loss in result quality should be achieved. Therefore, an algorithm has to support metamodel graph properties to utilize structural, type-based and linguistic information for a better quality and thus a smaller loss by partitioning. To summarize we derived the following requirements:

1. At most quadratic complexity,
2. Support of metamodel graph properties,
3. And a variable number of result partitions, i. e. a maximal size of partitions.

Subsequently, we will give an overview of graph partitioning and clustering algorithms and arrange them according to our requirements.

#### 6.2.1.2 Overview of graph partitioning and clustering algorithms

An overview of graph clustering and partitioning algorithms is given in Tab. 6.1. The table shows 18 algorithms organized in three groups and arranged according to our requirements. The first group depicts clustering algorithms, the second partitioning algorithms, and the third a combined algorithm. In the following we will explain our selection of the planar edge seperator (Planar t-Separator) algorithm [3] for metamodel matching.

The first requirement is given by quadratic runtime. The clustering approaches of Edge Betweenness from community detection [48], Markov chain-based clustering [26], Spectral Bisection [141], and Minimum Cutting Tree-based clustering [40] do not fulfil this requirement and are neglected. Next, the approaches Modularity [114], PNN [41], HCS [55], Kirchhoff equations [155], Planar Separator [96], and Multi Level clustering [22] do not support edge weights and thus typed graphs. Therefore, we do not consider these algorithms for partitioning.

Additionally, the approaches of Kernighan-Lin [78], Normalized Minimal Cut [137], Heuristic Optimization [5], hMetis [75], and Recursive Bisection [139] do not explicitly support a variable number of clusters or partitions. Although, an upper bound may be derived from a fixed number of clusters, the distribution of elements by the algorithms is done without consideration of structure, because the algorithms do not allow for size derivations as the graph-based partitioning. This behaviour leads to a non-consideration of these algorithms.

The remaining algorithms fulfilling our requirements are Local Betweenness [49], Chameleon [74], and the Planar Edge Separator [3]. We selected the planar edge separator because local betweenness shows a worse complexity and hence does not scale as well. Chameleon and the Planar Edge Separator are quite similar, because both follow the idea of initial splitting and remerging, but the authors of the planar edge separator argue their approach tends to perform better than hMetis [3]. Since Chameleon is an

| **Name** | **Complexity** | **Edge Weights** | **Variable no. of parts** | **Maximal size of parts** |
|---|---|---|---|---|
| **Clustering** | | | | |
| Modularity [114] | $O((n+m)n)$ | ✗ | ✓ | ✓ |
| PNN [41] | $O(n^2)$ | ✗ | ✓ | ✓ |
| Edge Betweenness [48] | $O(n^3)$ | ✗ | ✓ | ✓ |
| HCS [55] | $O(nm)$ | ✗ | ✓ | ✓ |
| Markov [26] | $O(n^3)$ | ✗ | ✓ | ✗ |
| Min. Cut. Tree [40] | $O(n^3)$ | ✓ | ✓ | ✓ |
| Normalized MinCut [137] | $O(n^2)$ | ✓ | ✗ | ✓ |
| Kirchhoff equations [155] | $O(n)$ | ✗ | ✗ | ✗ |
| Local Betweenness [49] | $O(m \log m)$ | ✓ | ✓ | ✓ |
| **Partitioning** | | | | |
| Kernighan-Lin [78] | $O(n^2 \log n)$ | ✓ | ✗ | ✓ |
| Heuristic Optim. [5] | n.a. | ✓ | ✗ | ✓ |
| Spectral Bisection [141] | $O(n^3)$ | ✗ | ✗ | ✓ |
| hMetis [75] | $O(n+m)$ | ✓ | ✗ | ✓ |
| Planar Separator [96] | $O(n)$ | ✗ | ✗ | ✓ |
| *Planar Edge Separator* [3] | $O(n^2)$ | ✓ | ✓ | ✓ |
| Recursive Bisection [139] | $O(n+m)$ | ✓ | ✗ | ✓ |
| Chameleon [74] | $O(n^2)$ | ✓ | ✓ | ✓ |
| **Hybrid** | | | | |
| Multilevel Clustering [22] | $O(n)$ | ✗ | ✗ | ✓ |

Table 6.1: Graph clustering and partitioning algorithms; ✓– requirement fulfilled, ✗– requirement not fulfilled, $n$ – no. of vertices, $m$ – no. of edges

additive extension of hMetis we conclude that the planar edge separator performs better.

However, for comparison we selected a representative algorithm for local betweenness from software clustering [15] and a density-based clustering algorithm [93] similar to Chameleon and compared them in context of a diploma thesis [59], concluding that indeed the PES performs best. In the following section we will explain our adoption of the PES as well as our extending merging phase based on coupling and cohesion.

### 6.2.2 Planar Edge Separator based partitioning

The main idea of the partitioning algorithm is to split a metamodel into subgraphs of similar size by removing vertices from the graph and calculating connected components[1]. The input of the algorithm is a maximal partition size or maximal number of partitions. The overall partitioning is achieved in three phases, partitioning, re-partitioning, and merging.

1. *Partitioning* First, an input metamodel graph is initially partitioned using a heuristic approach by removing vertices and their edges. These vertices and edges are selected based on levels of the metamodel graph. The partitions are calculated by the connected components of the vertices remaining after removal.

2. *Re-partitioning* Since some of the resulting partitions may exceed the size limit, they are re-partitioned in quadratic time while taking advantage of their planarity. Thereby again vertices are removed to recalculate connected components.

3. *Merging* Finally, all previously removed vertices that do not belong to any partition need to be merged with the previously calculated partitions to retain maximum structural information. There we propose to apply a merging of elements and partitions based on coupling and cohesion of partition pairs, for maximal structural information and thus matching result quality.

The three phases of our algorithm are given in Alg. 6.1 and are detailed in the following paragraphs. To illustrate the algorithm we use an extended version of the POS metamodel of Sect. 3.1.2 as shown in Fig. 6.2. It gives an example of a retail store as previously introduced with a class diagram on the left and the corresponding graph on the right side.

For the sake of simplicity the graph shows only classes, but attributes are represented in a similar way. Please note that class labels and relation types

[1]A connected component is a connected subgraph, i. e. each vertex is reachable from each vertex. One example algorithm for a connected component calculation is the Depth First Search (DFS).

are abbreviated. Based on this example a calculation of the partitioning is depicted in Fig. 6.3.

**Algorithm 6.1** Planar graph-based partitioning algorithm outline

```
Input: Graph $G(V, E)$, maximal partition size $w_{max} = tw(G)$
1. Partitioning
(1) Add virtual vertex $v_0$ to $G$ and connect all $v_i \in degree_{max}(G, k)$ with k such that all
    $v \in V$ are reachable.
(2) Compute SSSP tree $T$ rooted in $v_o$ w.r.t. $G$.
(3) Select a set of levels $L$ in $T$, using a heuristic restricted by $w_{max}$.
(4) Move vertices in $L$ into $V_{sep}$ and compute connected components $P_1, \dots P_m$ of graph $G$
    with $V \setminus V_{sep}$.
2. Re-partitioning
(5) Construct SSSP tree $T_j$ consistent with $T$ for partition $p_j$ with $w(p_j) > w_{max}$.
(6) Select levels $L_j$ with respect to $T_j$, whose removal partitions $p_j$ into components of
    weight $< w_{max}$.
(7) Insert vertices of $L_j$ into $V_{sep}$ and compute the connected components in $p_j$ with $V_j \setminus$
    $V_{sep}$.
3. Merging
(8) For each pair of partitions $(p_i, p_j)$ compute $coup(p_i, p_j)$ and $coh(p_i, p_j)$ and add them to
    the list of merging candidates iff $coup(p_i, p_j) > thres_{coup}$ and $coh(p_i, p_j) < thres_{coh}$.
(9) Select $(p_i, p_j)$ with maximal coup and merge to new partition $p_{ij}$, if $w(p_{ij}) \leq w_{max}$.
    Re-compute $coup(p_i, p_{ij})$ and $coh(p_i, p_{ij})$.
(10) Repeat from (8) as long as $(p_i, p_j)$ exists with $coup(p_i, p_j) > thres_{coup}$ and $w(p_{ij}) \leq$
    $w_{max}$.
```

**1. Partitioning** The first phase of the algorithm is dedicated to an initial partitioning of the input metamodel graph. The goal is to find a minimal set of vertices to be removed to achieve maximal sized partitions. The size is bounded by the input $w_{max}$. Thereby, we follow a heuristic level-based approach as given in [3], which reduces the partitioning problem to a Single Source Shortest Path (SSSP)[2] problem.

As given in Alg. 6.1 (1), a SSSP tree is rooted in a virtual element vertex $v_0$ that is required for a connected graph as a basis for the SSSP tree calculation[3]. As an extension of the original algorithm, we propose to connect $v_0$ to element vertices of a maximal degree until all elements are reachable. First, the element vertex of the maximal degree (maximal number of edges) is connected to $v_0$, as a result all vertices it connects to are also indirectly

[2]A SSSP algorithm computes the shortest paths from one vertex to all other vertices with respect to certain costs.

[3]The vertex $v_0$ is needed as the root package of a metamodel cannot be used for partitioning, since every vertex would be in the same level due to its direct reachability from the root package.

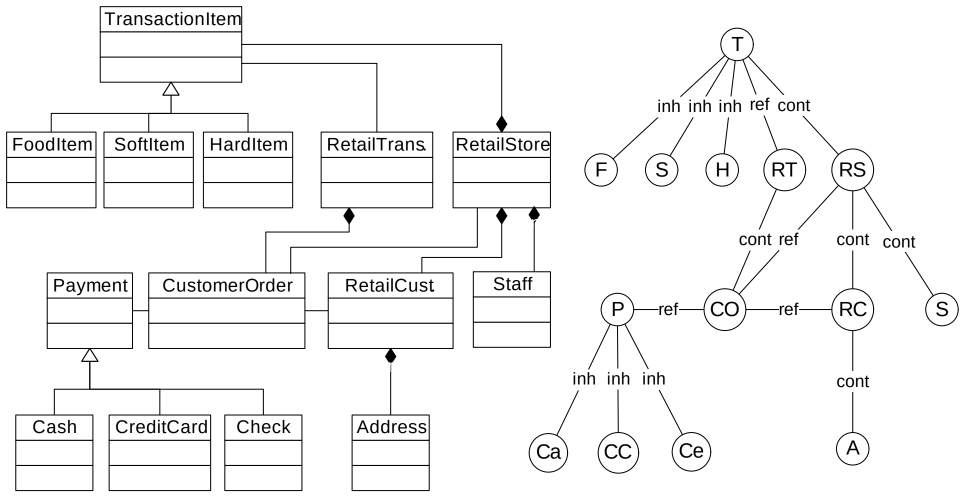


Figure 6.2: Example metamodel and the corresponding graph

connected to $v_0$. Next, the element vertex of the remaining unreachable element that has the maximal degree is selected and again connected to $v_0$. This procedure is repeated until all elements are reachable via $v_0$. We chose the maximal degree as criterion because then a minimal number of new edges to $v_0$ is created, since a maximal degree indicates a higher number of reachable vertices.

The vertex $v_0$ is the root of the SSSP tree calculation as in Alg. 6.1 (2), e. g. by using Dijkstra's algorithm [21]. The resulting SSSP tree is then arranged in levels where each level is a set of vertices that have the same distance to be reached. An example of a level graph is given in Fig. 6.3 (b).

(3) To select levels for removal we apply a heuristic as proposed in [3]. Outlining their approach, first a level graph is constructed (Fig. 6.3 (a) shows the levels and (b) the resulting level graph). That is a graph where each vertex $l_i$ represents a level $L_i$ and is connected to all levels with a higher distance. Each edge between two vertices $v_i$, $v_j$ gets costs assigned as follows:

$$cost(v_i, v_j) = cost(L(v_j) \setminus L(v_i)) + 2\lfloor \frac{2w(G_{i,j})}{w_{max}} \rfloor (d(v_{j-1}) - d(v_i)) \quad (6.1)$$

$L(v_{\{i,j\}})$ is a set of vertices that has the same distance as $v_x$ to $v_0$, thus the first part of Equation 6.1 depicts the cost for the removal of level $L(v_j)$. The second part considers the resulting weight of partitions when removing $L_j$. The sum of elements in levels between $L(v_i)$ and $L(v_j)$ is $w(G_{i,j})$, while $d(v_i), d(v_{j-1})$ represent the number of levels between $L(v_i)$ and $L(v_{j-1})$ respectively. These costs are used for a shortest path computation on the level

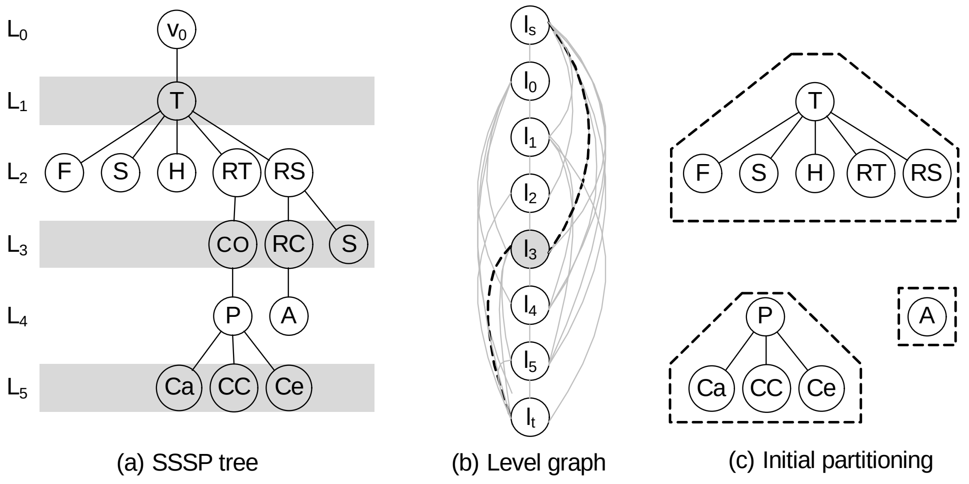


Figure 6.3: Example calculation for planar partitioning

graph. The resulting shortest path is a representation of levels to be removed, which have the lowest costs.

(4) The removal of levels from the graph is done by adding their vertices to the separator set $V_{sep}$. The corresponding edges of the removed vertices are also removed from the graph. The remaining vertices of the graph $G$ are calculated for their connected components and form the initial set of partitions as in Alg. 6.1 (8). Since a heuristic approach has been applied it can happen that some partitions exceed the upper bound $w_{max}$ and have to be re-partitioned.

An example of the initial partitioning phase is given in Fig. 6.3. First, the virtual root $v_0$ has been added to the metamodel graph and connected to the element vertex with the maximum degree that is *TransactionItem (vertex T)*. Since all elements are reachable no additional connection has to be introduced. Following the SSSP calculation the resulting tree is depicted in Fig. 6.3 (a). All resulting six levels are shown in our example labelled $L_0, \cdots, L_5$. These levels form the level graph with added source and target vertices $l_s, l_t$ as depicted in Fig. 6.3 (b). Each vertex represents a level of the SSSP tree and is connected to its succeeding levels. For our example the shortest path from $l_s$ to $l_t$ is via $l_3$, thus this level is selected for removal and the contained vertices and their edges are removed as depicted in Fig. 6.3 (c). The result is formed by calculating the connecting components of the remaining vertices leading to the three partitions.

**2. Re-partitioning** The optional re-partitioning phase is applied to all partitions $P$ that exceed the size limit $w_{max}$ as given in Alg. 6.1 (5–7). Since it is a complex calculation and identical to the one proposed in [3] (pp. 5–9) we refrain from detailing it. The phase is optional but some partitions may violate the size limit.

A SSSP tree $T_j$ for $p_j$ is calculated, which has to be consistent with the original tree $T$. A tree is called consistent if all vertex distances of $T_j$ have at most the distances of the SSSP tree $T$ of phase 1.

Of this SSSP tree, two levels can be removed in a way that the resulting three partitions are at most half of the original weight using the approach given in [3]. These two levels are removed by adding their vertices to $V_{sep}$ and the connected components are calculated. If necessary a set of fundamental cycles[4] are removed. The removal guarantees resulting partitions of sufficient weight $\leq w_{max}$ and hence size.

Having ensured the maximal size of all partitions $p \in P$ the remaining vertices in $V_{sep}$ are still unassigned and thus not part of any partition. They are merged with the partitions calculated w.r.t. the size $w_{max}$ as described in the following phase.

**3. Merging** The previous partitioning and re-partitioning produces two outputs: a set of partitions $P$ and the elements removed during the phases $V_{sep}$. Since the elements in $V_{sep}$ are not part of any partition they need to be merged with the previously calculated partitions. In contrast to the simple random merge strategy of [3], we propose a merge strategy based on structure, thus being structure-preserving. To ensure the matching of all elements, we propose to add the remaining elements of the separator $V_{sep}$ to $P$ as partitions of the size of one element. We select partitions to be merged on two measurements: coupling and cohesion.

We propose to perform the merging on a weighted graph allowing for metamodel specifics. Thereby, we define the edge weights of the input metamodel graph in accordance with the density-based clustering approach [93], i. e. attribute edges have a weight of 5, containment/aggregation of 4, associations of 2, inheritance has a weight of 1. The weight assignment follows the rationale of importance that means a higher weight indicates a higher importance of a relation.

First, the merging phase as given in Alg. 6.1 (8–10) calculates coupling and cohesion for all pairs of partitions. Inspired by [74] we define coupling and cohesion as follows:

$$coup(p_i, p_j) = \frac{w(E_{\{p_i,p_j\}})}{\frac{w(E_{p_i})+w(E_{p_j})}{2}} \tag{6.2}$$

$$coh(p_i, p_j) = \frac{w_{avg}(E_{\{p_i,p_j\}})}{\frac{|p_i|}{|p_i|+|p_j|} \cdot w_{avg}(E_{p_i}) + \frac{|p_j|}{|p_i|+|p_j|} \cdot w_{avg}(E_{p_j})} \tag{6.3}$$

[4]A fundamental cycle is a path inside a tree with the same start and end vertex containing at most one non-tree edge

Equation 6.2 defines coupling as the ratio between the sum of weights of edges connecting both partitions ($w(E_{\{p_i,p_j\}})$) and the average of the sum of edge weights $\frac{w(E_{p_i})+w(E_{p_j})}{2}$ in both partitions $p_i, p_j$. Complementarily, cohesion is defined by taking the relative partition size into account. Cohesion weights the average edge weight sum of each partition ($w_{avg}(E_{p_i})$) by the relative size of a partition ($\frac{|p_i|}{|p_i|+|p_j|}$), with $|p_i|$ as the number of elements in $p_i$ w.r.t. both partitions. We chose both measurements because coupling allows to rank source and target partitions based on their connectivity where cohesion allows to preserve partitions which consist of closely related elements, instead of merging them and thus adding misleading information for matching.

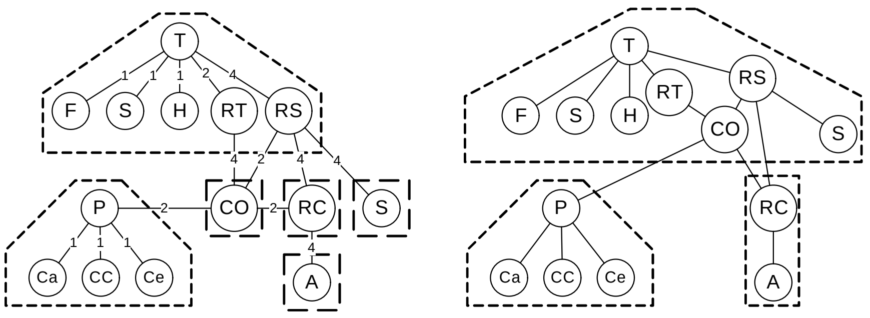
In case of two partitions with one element each and one connecting edge we propose to set coupling to the weight of the connecting edge and cohesion to one, thus ensuring a preferred merging of single element partitions.

Beginning with the pair of partitions with maximal coupling the partitions are merged as given in Alg. 6.1, phase 3 (9). Thereby, a pair $p_i, p_j$ is merged if the cohesion fulfils $coh > thres_{coh}$. We chose $thres_{coh} = \frac{4}{5}$ because it captures the borderline case of two partitions with connected elements. That is the inner connection of two partitions has the maximum weight 5 for attribute relations and should not be merged if there is a weaker connection between both partitions, i. e. 4 for containment relations. Additionally, the maximum size restriction has to be fulfilled. The coupling and cohesion values have to be calculated for the new merged partition $p_{new}$ and all partitions connected to $p_i, p_j$.

The merging of partitions of maximal coupling is repeated as given in Step (10) until no pair can be found which either has a sufficient cohesion or which cannot be merged without violating the upper bound $w_{max}$. The final output of the algorithm is the set of partitions $P$ which has been created in the merging phase.

Figure 6.4 depicts an example for the merging phase. On the left (a) the partitions resulting from the previous initial partitioning are depicted as dashed line polygons, the numbers of the lines represent the weights of relations as defined by us before. The coupling of $P, CO$ is calculated by the weight of edges between them $w(E_{\{P,CO\}}) = 2$ and the average of their own (inner) edges' weights, for the partition including $P$ that is $w(E_P) = 1 + 1 + 1 = 3$ and $w(E_{CA}) = 0$. Consequently, $coup(P, CA) = \frac{2}{\frac{3+0}{2}} = \frac{4}{3}$. Calculating the coupling of the other pairs yields the following numbers $(RT, CO) = \frac{4}{3}, (RC, A) = 4, (CO, RC) = 2, (RS, S) = \frac{8}{9}, (RS, RC) = \frac{8}{9}$. Please note that the pair $(RS, CO)$ is identical to the partition pair $(RT, CO)$. Resulting from this the partitions *RC* and *A* are merged, since they have the highest coupling.

In the next step considering the recalculated coupling and cohesion values, the partitions *CO* and *RT* are merged. The partitions $CO$ and $(RC, A)$



(a) Before merging (b) After merging

Figure 6.4: Example of the partition merging phase

are not merged because of their cohesion. The cohesion $coh(CO, RT)$ is calculated with $w_{avg}(CO) = 0$ because $CO$ has no edge and $w_{avg}(RC, A) = 4$ because $(RC, A)$ has only one edge. The cohesion is $\frac{2}{\frac{1}{3}\cdot 0+\frac{2}{3}\cdot 4} = \frac{3}{4}$ and does not exceed the threshold of $\frac{4}{5}$. Finally, $(RS, S)$ are merged while the other partitions do not satisfy the cohesion threshold. Since then no more merge partners are available, the final output is exactly as depicted in Fig. 6.4 (b).

We have presented our planar graph-based partitioning, which splits a metamodel into subgraphs of similar size by optimizing structural information. The optimization is achieved by our proposed merging of partitions based on coupling and cohesion, thus leading to better matching results by structural matchers. Finally, our partitioning reduces the memory consumption of a matching system and allows for distributed matching, because it produces enclosed matching tasks. However, it still does not tackle the problem of runtime, especially on a local machine, which occurs due to pair-wise comparison of all partition elements. We address this in the following section.

## 6.3 Assignment of Partitions for Matching

Having tackled the memory consumption of a matching system the runtime still remains an issue and even increases by the partition calculation. Considering a pair-wise comparison of all partitions of a source and target metamodel the result is the cartesian product and thus matching of all source with all target elements. Figure 6.5 depicts an example of pair-wise assignment of all partitions; the source partitions are represented by grey circles, the target partitions by white ones, the assignment and consequently the pairs for matching are represented by arrows. The problem is now to reduce the number of comparisons with a minimal loss in matching quality. That means an algorithm is needed which determines relevant partitions and their assignment to be matched without performing the actual match-

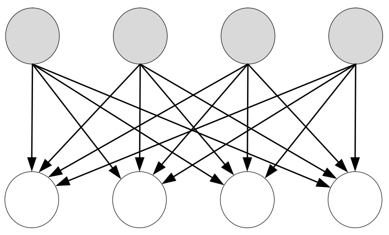


Figure 6.5: Partition matching without assignment

ing. This process is called partition assignment and is based on matching those partitions which have the highest similarity. The similarity is calculated using partition representatives instead of pair-wise element comparison to reduce the computational overhead.

In this section we study and compare four partition assignment algorithms. These they are:

- Threshold-based and quantile-based assignment, selecting a subset for matching,
- Hungarian and generalized assignment, aiming at optimal one-to-one or one-to-many partition assignments respectively.

First, a similarity measurement for partitions has to be defined as discussed in the following subsection. Then, we can apply and discuss the four algorithms for partition assignment.

### 6.3.1 Partition similarity

The first step towards the partition assignment problem is to obtain similarity values for each partition pair. These values can be calculated in various ways by applying arbitrary matching techniques, e. g. in [156]. However, if such techniques are applied on every element of a partition the final result is again the cartesian product of all elements, which is undesirable because of the computational effort.

Therefore, we propose to first select representatives for each partition. Then these representatives are compared pair-wise using structural and linguistic information, leading to source and target partition similarities. Since the selection itself should not introduce additional overhead we propose to base the representative selection on the $k$-max degree as introduced by us in Sect. 5.1.4, where $k$ represents the number of representatives per partition. The selection is done in linear time and uses the $k$ elements with the maximum degree, i. e. the maximal number of neighbours.

The partition similarity is obtained by the similarity measures introduced for the graph edit distance matcher in Sect. 5.1. There we defined linguistic and structural similarity.

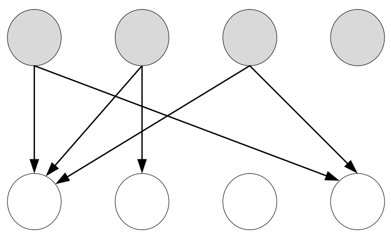


Figure 6.6: Partition matching with threshold-based assignment

**Linguistic similarity** The linguistic similarity is defined by us as the distance between the labels of two given elements. Thereby, we make use of a name matcher, e. g. a tri-gram similarity.

**Structural similarity** We define the structural similarity as the ratio of edges of two given elements. Thereby, we consider containment, attribute, and inheritance edges and average the results. Recapitulating it is defined for two given elements $v_s$ and $v_t$ as:

$$struct(v_s, v_t) = \frac{1}{2} \cdot attr(v_s, v_t) + \frac{1}{2} \cdot ref(v_s, v_t) \tag{6.4}$$

The structural and linguistic similarity can be aggregated for a cluster similarity calculation or be used separately. We demonstrated in the context of a master thesis [59] that using structural similarity is superior to linguistic, because it yields the same matching result quality but a better runtime.

### 6.3.2 Assignment algorithms

In the following sections we will describe two straight-forward approaches: threshold-based and quantile-based assignment. Since both approaches use the calculated similarity values to exclude partition assignments they may miss partition pairs to be selected for matching. Therefore, we also present two approaches mapping the assignment problem on an optimization problem aiming at one-to-one or one-to-many assignments. These approaches are called Hungarian and generalized assignment.

#### 6.3.2.1 Threshold-based assignment

A high similarity of a given partition pair indicates a large number of similar elements in both partitions. Therefore, selecting pairs of partitions with a high similarity should result in a large number of matching elements. The rationale behind defining a threshold $thres$, as for instance in the matching system Falcon-AO [156], is to select only those partition pairs that have a similarity exceeding $thres$. This can be formulated as in (6.5).

$$f(thres) = \{(p_i^s, p_j^t) | sim(p_i^s, p_j^t) \geq thres\} \tag{6.5}$$

The cause problem of this approach is the definition of the threshold itself. A threshold $thres$ largely depends on the given scenario and varies as shown in our evaluation. For instance, consider an example of 4 source and 4 target partitions with similarity values between the partition pairs of 0.5 and 0.8. A threshold of 0.9 would fail to select any pair at all, even though it may have worked for previous scenarios. In contrast, a threshold of 0.4 would select every pair for matching. Therefore, a preferable (average) threshold that works best for any scenario cannot be given.

The threshold-based assignment shows a computational complexity of $O(n^2)$ where $n$ is the maximum of source and target partition count, because for a given threshold each pair needs to be checked for its similarity.

Figure 6.6 depicts an example result for threshold-based assignment. In this example a subset of $6$ pairs is selected for matching, but two partitions have no match partner, which shows the potential problems using a threshold-based assignment. Please note, that this example serves as a comparative example for the non-assignment given in Fig. 6.5.

#### 6.3.2.2 Quantile-based assignment

We propose quantile-based assignment to overcome the scenario dependency of threshold-based assignment. A quantile $q$ describes a fraction of the partition pairs to match, e.g. the quantile $q = 0.5$ selects half of all partition pairs with the highest similarity. In case the number of source partitions is denoted as $|P_s| = m$ and the number of target partitions as $|P_t| = n$, then the number of selected pairs based on the quantile is $\lceil q \cdot n \cdot m \rceil$ and these pairs can be defined as:

$$\begin{aligned} f(q) = Q := \{(p_i^s, p_j^t) | p_i^s \in P_s \wedge p_j^t \in P_t \wedge |Q| = q \cdot |P_s| \cdot |P_t| \wedge \\ \forall (p_i^s, p_j^t) : sim(p_i^s, p_j^t) \geq sim(p_k^s, p_l^t), (p_k^s, p_l^t) \in (P_s \times P_t) \setminus Q\} \end{aligned} \tag{6.6}$$

Unlike the threshold-based assignment the quantile-based approach allows to predict the number of partition pairs selected and thereby can prevent having none or all partition pairs selected for matching. This produces a better result quality (see our evaluation in Sect. 7.5.1) and limits the influence of a specific scenario.

The quantile-based assignment shows a computational complexity of $O(n^2 logn)$ with $n$ as the partition count, because all partition pairs need to be sorted according to their similarity, for instance by Quicksort ($O(nlogn)$) and then selected w.r.t. a given quantile.

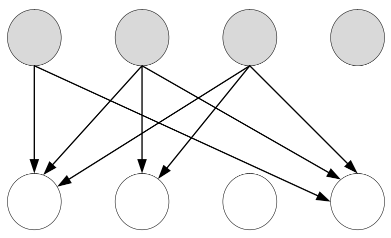


Figure 6.7: Partition matching with quantile-based assignment

In Fig. 6.7 an example result for quantile-based assignment with $q = 0.5$ is depicted. As in threshold-based assignment a subset of partitions to be matched is selected, but in contrast to threshold-based assignment more partitions get selected, which are $8$ that means $50\%$ of all possible pairs $(16)$. Still, two partitions are not assigned and consequently will not be matched.

### 6.3.2.3 Hungarian assignment

Since the partition assignment is closely related to the Knapsack problem [105] we propose to apply two algorithms from this area. The Knapsack problem deals with the problem of an optimal distribution of $n$ elements to $m$ containers. The Hungarian algorithm [85] proposed 1955 by three Hungarians, is one solution for the Knapsack problem. The partition assignment problem is the same problem dealing with an optimal distribution of $n$ partitions to $m$ partitions.

The assignment problem originally underlying the Hungarian algorithm tries to identify the set of optimal assignments of workers to jobs. Mapped on the partition assignment problem workers represent the source partitions where the jobs are represented by the target partitions. Thereby, a set of jobs $P_s$ (source partitions) and a set of workers $P_t$ (target partitions) are assigned in a one-to-one manner. That means only one job is assigned to one worker and vice versa, thus they form exact one-to-one assignments. The assignments are represented in a boolean matrix $M = \{m_{ij}\}$ with i as the worker and j as the job, with $m_{ij} = 1$ if a job is performed by a corresponding worker and else 0. A combination of a job and a worker also has an associated cost value represented in a cost matrix $C$ with each matrix element $c_{ij} \in [0, 1]$ . The optimal solution assigns each worker a job while minimizing the costs as defined in the following equation.

$$\begin{aligned} &\text{minimize} \sum_{i=1}^{|P_s|} \sum_{j=1}^{|P_t|} c_{ij} \cdot m_{ij} \\ &\text{subject to} \sum_{i=1}^{|P_s|} m_{ij} = 1, j \in 1 \dots |P_t| \wedge \sum_{j=1}^{|P_t|} m_{ij} = 1, i \in 1 \dots |P_s| \end{aligned} \tag{6.7}$$

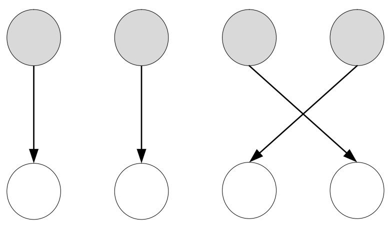


Figure 6.8: Partition matching with Hungarian assignment

We propose to adapt the solution for this problem for partition assignment in a one-to-one manner. That means the Hungarian algorithm can be used to identify one-to-one partition pairs with an optimal overall similarity. This is done by defining the costs as $c_{ij} = 1 - sim_{ij}$, because the costs depict the distance between two partitions. The main idea of the algorithm is to subtract from each row and column of the cost matrix the minimal value. By subtracting the row or column minimum the position of the minimum leads to at least one zero in each row and column. Then the algorithm tries to mark exactly $|P_t|$ rows or columns in such a way that all zeros are covered. If this procedure fails it again subtracts the minimum for all uncovered zeros. Thereby, the algorithm shows a complexity of $O(n^3)$. Since, we adopted the algorithm unchanged we confer for details to [85].

Figure 6.8 depicts an example output of the Hungarian assignment. As described, one-to-one assignments are produced with an overall optimal similarity. Unfortunately, multiple assignments of source to target partitions are not identified, since every source gets exactly one target assigned. The problem of one-to-one assignments becomes more distinct in case of a count mismatch between the source and target partitions. For instance, in case of 5 source and 100 target partitions only 5 assignments can be computed. The low number of assignments may lead to a decrease in matching result quality. Therefore, we also investigate the generalized assignment as described in the following.

#### 6.3.2.4 Generalized assignment

The Generalized Assignment [105] problem also tackles the assignment of a set of source and target elements w.r.t. costs. In contrast to the Hungarian algorithm it relaxes the condition of one-to-one assignments to one-to-many.

Generalized Assignment also deals with the Knapsack problem and an element to bag distribution. Therefore, a profit matrix $P = p_{ij}$ is defined, representing the profit of assigning an element to a bag. For our partition assignment problem that is the assignment of a source partition to target partitions and thus their representative similarity $sim_{ij} = sim(p_i^s, p_j^t)$. In addition, every target partition $p_j^t \in P_t$ gets a capacity $c_j$ (similar to a con-

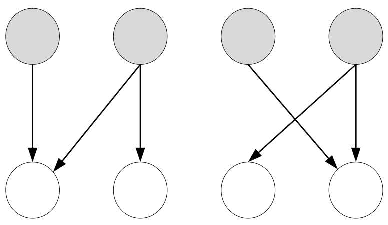


Figure 6.9: Partition matching with generalized assignment

tainer). The capacity denotes the maximal number of weight assignable to $p^t_j$. Consequently, every pair of source partition $p^s_i \in P_s$ and target partition $p^t_j$ is assigned a weight $w_{ij}$.

The weight and capacity determine the degree of assignability between two partitions. We chose to use the number of matching partition representatives for two partitions $p_s, p_t$ as weight. This number contains the representatives $e$ with similarity values exceeding a certain threshold $x$. The number of representatives considered and chosen by their degree is a user-defined parameter. We note the weight definition as follows:

$$w_{ij} = w(p^s_i, p^t_j) = |\{(e_s, e_t)|sim(e_s, e_t) > x, e_s \in p^s_i, e_t \in p^t_j\}| \quad (6.8)$$

We define the capacity as the union of all possible weights because this represents the maximal number of assignments possible. That means, $c_j = |\bigcup_{i,j}\{(e_s, e_t)|sim(e_s, e_t) > x, e_s \in p^s_i, e_t \in p^t_j\}|$. The following equation summarizes the optimization problem to be solved.

$$\begin{aligned} &\text{maximize} \sum_{i=1}^{|P_s|}\sum_{j=1}^{|P_t|} sim_{ij}x_{ij} \\ &\text{subject to} \sum_{j=1}^{|P_t|} w_{ij}x_{ij} < c_j \wedge \sum_{i=1}^{|P_s|} x_{ij} = 1 \\ &x_{ij} \in \{0,1\}, i \in 1\ldots|P_s|, j \in 1\ldots|P_t| \end{aligned} \quad (6.9)$$

Unfortunately, the Knapsack problem is NP-complete and not decidable [105]. Therefore, several algorithms have been proposed to find an approximate solution to the problem as presented in [13]. Martello and Tooth [104] proposed an algorithm with an average deviation of 0.1 % compared to the optimal solution. Since we implemented the algorithm unchanged we only give a short outline.

The algorithm consists of two phases: an initial assignment phase and an optimization phase. The initial phase is executed for each of the measures $p_{ij}, \frac{p_{ij}}{w_{ij}}, -w_{ij}, -\frac{w_{ij}}{c_j}$ choosing assignments in the way that the minimum of the measures is assigned to the maximum of the measures. The procedure is done for all elements and values choosing the solution with the highest profit. Subsequently, the optimization phase tries to swap elements to improve the profit under the capacity restriction.

Figure 6.9 depicts an example output for the generalized assignment solution of the partition assignment problem. It shows that every source partition gets at least one target partition assigned for matching and that multiple assignments for one source partition are part of the output. However, the weight calculation and thus the capacity calculation need the number of representatives and the definition of a threshold which introduces another parameter. The algorithm also suffers from a decrease of the fraction of pairs selected with increasing input size. The more partitions are part of the input the smaller the fraction of all possible pairs selected for matching. The fewer pairs selected for matching the fewer elements are matched which potentially decreases the result quality.

### 6.3.3 Comparison

The four assignment approaches are different in their nature and each shows advantages and disadvantages. Therefore, we have arranged the approaches in Tab. 6.2. The first column shows the approaches and the next columns list the corresponding advantages and disadvantages. The threshold-based assignment is the simplest approach and allows for many-to-many mappings. As discussed before it suffers from the fixed number as threshold and thus is scenario specific. Besides the major drawback of scenario dependence also the coverage, that is the fraction of elements of a metamodel considered for matching, is unknown. As shown in the example in Fig. 6.6 the threshold-based assignment may skip partitions for matching and thus lower the coverage. Coverage is the share of assigned partition pairs to all possible pairs, i. e. the cartesian product.

Quantile-based assignment also allows for many-to-many mappings and shows stable results in contrast to threshold-based assignment. Since a fraction of all possible pairs is selected the number is predictable. However, quantile still suffers from the problem of coverage, because partitions may be skipped for matching, thus the coverage is unknown.

Interpreting assignment as an optimization problem the approach of Hungarian assignment produces one-to-one mappings for all partitions and thus it is complete. That means it shows a complete coverage, because every partition will be considered for matching. The strength of the Hungarian is also its weakness because it does not allow for one-to-many assignments. It also shows cubic complexity ($O(n^3)$) which introduces computational overhead. Since the Hungarian algorithm selects optimal one-to-one pairs, the overall number of pairs is lower than for quantile (equal to the number of source or target partitions).

The generalized assignment approximation tries to overcome some of the limitations of the Hungarian algorithm. It allows for one-to-many map-

| Assignment | Complexity | Advantages | Disadvantages |
|---|---|---|---|
| Threshold | $O(n^2)$ | $n:m$ mappings | manual threshold, scenario dependent, unknown coverage |
| Quantile | $O(n^2 logn)$ | $n:m$ mappings, selected pairs predictable | unknown coverage |
| Hungarian [85] | $O(n^3)$ | $1:1$ mappings, complete coverage | $O(n^3)$, no $n:1$ mappings |
| Generalized [104] | $O(n^2 logn)$ | $1:n$ mappings, complete coverage | uses internal threshold |

Table 6.2: Advantages and disadvantages of the four assignment approaches

pings while having a quadratic complexity. However, it still shows a small number of pairs being selected for matching which may reduce matching quality. Additionally, the generalized assignment relies on an internal threshold (the capacity) which needs to be defined on average scenarios.

As discussed all algorithms show advantages as well as disadvantages. Therefore, we will examine the algorithms in our evaluation to derive recommendations for the usage of partition assignment approaches.

## 6.4 Summary

In order to solve the memory problems in the context of large-scale metamodel matching we proposed a planar graph-based partitioning and partition-based matching process. To reduce runtime we also considered the partition assignment problem. Thereby, our novel approach determines and compares partition representatives and based on their similarity selects partitions for matching. We studied four solutions, two of them mapping the assignment on an optimization problem. These contributions are summarized in Tab. 6.4 and detailed as follows.

**Planar graph-based partitioning** We have presented a planar partitioning approach to cope with the demands of large-scale metamodel matching. First, we deduced a requirement driven analysis of existing partitioning and clustering algorithms. We concluded with the selection of planar graph-based partitioning [3], also refered to as PES, mainly because of its quadratic runtime and support for metamodel graphs. We adopted the three-

| Concept | Contributions |
|---|---|
| Planar graph-based partitioning | • Planar graph-based partitioning for matching with a partition-based matching process<br>• Definition of seed vertex for partitioning based on $k$-max degree<br>• New partition merging phase based on coupling and cohesion |
| Partition assignment | • Partition similarity calculation based on $k$-max degree representatives<br>• Analysis and comparison of four assignment approaches<br>• Mapping of the partition assignment problem on the generalized assignment algorithm |

Table 6.3: Contributions of graph-based partitioning and assignment

phase approach of planar partitioning, which splits an input metamodel into partitions of similar size by an initial partitioning utilizing our $k$-max degree approach and re-partitioning by removing elements. Finally, these elements and partitions are merged as proposed by us using coupling and cohesion to optimize the amount of structural information per partition.

**Partition assignment** To reduce the number of comparisons between partitions we proposed solutions for the partition assignment problem. There, partitions get assigned in pairs for matching based on their similarity, preferring pairs with high similarity for increased result quality. For an efficient partition similarity calculation we propose to apply our $k$-max degree approach for selecting partition representatives. These representatives are used for partition similarity calculation. For the selection of partitions to be matched we investigated four partition assignment approaches: threshold, quantile, Hungarian and generalized assignment.

Based on an initial similarity between partitions, a selection of pairs has to be made. Thereby, threshold and quantile allow for many-to-many partition assignments by either selecting pairs above a certain threshold or a fraction of possible pairs. In contrast, Hungarian and Generalized Assignment try to solve the assignment as an optimization problem. They produce either one-to-one (Hungarian) or one-to-many assignments (Generalized Assignment). These approaches will be compared in our evaluation.

# Chapter 7

# Evaluation

In Chap. 5 we presented a planar graph edit distance matcher and graph mining-based matching to improve matching result quality. To tackle the runtime and memory issues we proposed in Chap. 6 planar graph-based partitioning and discussed four partition assignment algorithms. This evaluation chapter will answer the question: To which degree did we achieve our goals to improve and support large-scale metamodel matching?

Therefore, we present the fourth contribution of our work: a systematic evaluation based on existing real-world large-scale mappings from the MDA community and SAP business scenarios. These mappings are compared to the automatic results from our validating matching system. This matching system incorporates our matchers and the proposed partitioning. In detail, we first present our evaluation strategy and describe the data sets used. We distinguish academic data sets from real-world business data and will show that especially the real-world data fulfils the requirements of a hard and diverse data set. We then describe our matching framework MatchBox, which serves as the implementation basis for our evaluation. Subsequently, we introduce the measurements used. We then evaluate our matchers and the partitioning in terms of correctness and completeness as well as memory consumption and runtime behaviour. Finally, we summarize and discuss our results by presenting the applicability and limitations of our proposed solutions.

## 7.1 Evaluation strategy

In order to study the quality of a matching system, the quality of the mappings calculated has to be assessed. This assessment is done by comparing the mappings calculated with existing mappings, so-called gold-standards, for two given metamodels. Based on this *correctness* and *completeness* and their *harmonic mean* can be derived. Those measures are also widely used in matching system evaluations, e. g. in [126, 37, 33, 38].

Based on this approach the main goal of our evaluation is to demonstrate to which degree our solution meets the requirements and corresponding goals. We define the following success criteria based on our requirements. The first goal is *to increase the correctness and completeness of matching results (G1)*, the second to *support matching of large-scale metamodels (G2)*. For the first of our goals *G1* we derived the following questions to be answered by our evaluation:

- To which degree does our planar graph edit distance matcher improve result quality?
- To which degree does our design pattern and redundancy matcher improve result quality?

The derived success criterion for our matchers is an increase in result quality. The methodology we apply is to compare our matchers to a baseline. The baseline is a matching system with state of the art matching techniques. Thereby, we observe and interpret resulting changes in terms of correctness and completeness of the matching results.

In order to assess the goal *G2* regarding our graph-based partitioning we formulate the following questions:

- Which partition size is the optimal choice w.r.t. to maximal result quality?
- To which degree does our planar partitioning reduce memory consumption?
- Which assignment algorithm should be used for runtime reduction while minimizing the loss in result quality?

The resulting success criterion is a decrease in memory and runtime with a minimal loss in result quality. The methodology we follow to answer these questions is to compare our matching system without (baseline) and with our partitioning algorithm. Thereby, we investigate memory consumption and runtime as well as changes in correctness and completeness to determine the trade-off between quality and scalability.

The main challenge of a matching system's evaluation is the test data, which means the gold-standards. Since we developed generic concepts we investigated several data sets from different technical spaces to apply our concepts. First, we extracted gold-standards from model transformations of the ATL-zoo as proposed by us in [151]. Since this data set is of academic nature we also investigated mappings available within SAP. Thereby, we discovered real-world message mappings between business schemas which serve as the second data set. In the following we will first describe our matching system's implementation and subsequently our data sets.

## 7.2 Evaluation framework: MatchBox

In this section we will introduce our evaluation matching system. Basically, it takes two metamodels as input and creates a mapping, i. e. correspondences between model elements, as output. In order to create mappings we use a matcher framework that forms the basis for combining results of different metamodel matching techniques. For this purpose, we adopted and extended a matcher combination approach as proposed by us in [146, 147].

We have chosen the *SAP Auto Mapping Core* (AMC), which is an implementation inspired by COMA++ [25], a schema matching framework. In contrast to COMA++, the AMC consists of a set of matchers operating on trees. It incorporates schema indexing techniques for industrial applications, whereas COMA++ is an academic prototype operating on a directed acyclic graph (closest to a tree).

The evaluation was performed on a laptop running Java. The laptop was running Java 1.6.0.22 64-bit on 4 Intel i5 cores with 2.4 GHz each. The main memory was 4 GB of which 2 GB had been assigned to Java. The operating system was Windows 7 on 64 bit.

In the following we will explain MatchBox's architecture and its components. Afterwards, we outline the matching algorithms implemented, demonstrating how they are applied to metamodels. Finally, we describe the combination of the different matchers' results by an aggregation and selection leading to a creation of mappings.

### 7.2.1 Processing steps and architecture

MatchBox is built around an exchangeable matching core, enriching it with a graph model and functionality for metamodel import, similar to a traditional matching system as described in Chap. 2. In order to create mapping results several steps have to be performed, as outlined in Fig. 7.1:

1. Importing metamodels into the internal data model of MatchBox,
2. Applying matchers to obtain similarity values,
3. Combining similarity values of different matchers (aggregation and selection) and creating a mapping.

These steps are in detail: (1) The metamodels have to be transformed into the internal graph model of MatchBox. This step is necessary to apply generic matching techniques, e. g. independent from the technical space or level of abstraction. Having transformed the metamodels, several matchers can be applied, each leading to separate results for two metamodel elements (2). The system can be configured by choosing which matcher should be involved in the matching process. Each matcher places its results into a matrix

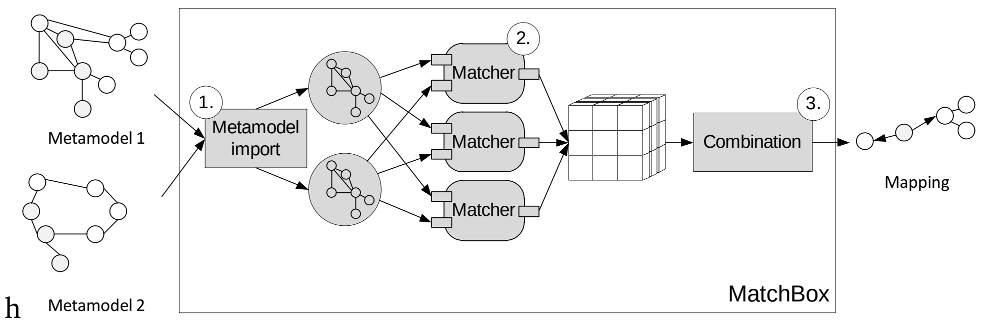


Figure 7.1: Processing of our metamodel matcher combination framework MatchBox [151]

containing the similarity values for all source/target element combinations. These matcher result matrices are arranged in a cube, which needs to be aggregated in order to select the results for a mapping. This is done in the third step (3) to form an aggregation matrix (e. g. by calculating the average). The entries in the matrix are similarity values for each pair of source and target elements. These values are filtered using a selection, e. g. by selecting all elements exceeding a certain threshold. Finally, the selected entries are used to create a mapping.

### 7.2.2 Matching techniques

The matchers of the core operate on the internal data model. Each matcher takes two elements as input and produces their similarity value as output.

Adopting the SAP AMC framework, we applied a set of most common matchers, namely: *name matcher*, *name path matcher*, *parent matcher*, *children matcher*, *sibling matcher*, *leaf matcher* and *data type matcher*. The concepts of the matchers implemented are described in the following.

**Name matcher** This matcher targets the linguistic similarity of metamodel elements. It splits given labels into tokens following a case-sensitive approach. Afterwards, for each token a similarity based on trigrams is computed. The trigram approach determines the total count of equal character sequences of size three (trigram) and finally compares them to the overall number of trigrams. Alternativly, a string edit distance based on Levenshtein [153] can be used.

**Name path matcher** This matcher performs a name matching on the containment path of an element. Hence, it helps to distinguish sublevel-domains in a structured containment tree even if leaf nodes do have equal names. Essentially, a name path is a concatenation of all elements along a containment path for a specific element. For matching the name matcher is applied on both name paths, thereby separators are omitted.

**Parent matcher** This matcher follows the rationale that having similar parents indicates a similarity of elements. The parent matcher computes the similarity of a source and target element by applying a specific matcher (e.g. the name matcher) to the source's and target's parents and returns the similarity calculated.

**Children matcher** The children matcher follows the rationale that having similar child elements implies a similarity of the parent elements. This matcher uses any matcher to calculate an initial similarity, for the implementation we chose the leaf matcher since this matcher shows the best results. The children matcher evaluates the set of available children for a given source and target node. Comparing both sets by applying the leaf matcher leads to a set of similarities which are combined using the average strategy.

**Sibling matcher** The sibling matcher follows an approach similar to the children matcher. It is based on the idea that a source element which has siblings with a certain similarity to the siblings of a given target element indicates a specific similarity of both elements. As in the children matcher, any matcher can be used for the calculation of similarity values for the siblings. In our implementation, we again chose the leaf matcher. The results of the separate matching between the different siblings are stored in a set. Finally, the set is combined as in the children matcher using the average of all values.

**Leaf matcher** This matcher computes a similarity based on similar leaf children. Thereby, the subtree beneath an element is traversed and all elements without children (leaves) are collected. This set of leaves corresponding to a source element is compared to the set belonging to a target element by applying the name matcher and aggregating the single results for the source and target element. These aggregated values are aggregated again using the average strategy.

**Data type matcher** The data type matcher uses a static data type conversion table. In contrast to the type system provided by XML, metamodels and in particular EMF allow a broader range of types. For example, EMF allows defining data types based on Java classes. We extended the data type matcher by conversion values for metamodel types and a comparison of instance classes. For instance, comparing two attributes one of type *EFloat* the other of type *EInt*, the data type matcher evaluates their data types, performs a look-up on its type table and returns a similarity of 0.6.

| Name matcher | A | B | C |
|---|---|---|---|
| X | 0.80 | 0.50 | 0.11 |
| Y | 0.44 | 0.66 | 0.08 |
| Z | 0.10 | 0.09 | 0.56 |

| Leaf matcher | A | B | C |
|---|---|---|---|
| X | 0.7 | 0.0 | 0.0 |
| Y | 0.0 | 0.47 | 0.17 |
| Z | 0.0 | 0.10 | 0.30 |

(i) Matcher results

| Result | A | B | C |
|---|---|---|---|
| X | 0.8 | 0.5 | 0.11 |
| Y | 0.44 | 0.66 | 0.17 |
| Z | 0.10 | 0.10 | 0.56 |

(ii) Aggregation (max)

| Source | Target | Value |
|---|---|---|
| X,Y | A | 0.7, 0.6 |
| Y | B | 0.66 |
| Z | C | 0.56 |

(iii) Selection (treshold > 0.4)

Figure 7.2: Example of aggregation and selection of matcher results

### 7.2.3 Parameters and configuration

Following the AMC concept, each matcher produces an output result in the form of a matrix. This matrix orders the source and target elements along the X and Y axis respectively. The cells are filled with similarity values between 0 and 1. All similarity matrices are arranged along the matcher types (Z axis) resulting in a cube. An example for a similarity matrix is given in Fig. 7.2, part (i).

In order to combine the results obtained, the combination component supports different strategies adopted from the AMC similar to the ones proposed in [8] or [25]. The strategies are *aggregation*, *selection*, *direction* and a *combination* thereof. The aggregation reduces the similarity cube to a matrix, by aggregating all matcher result matrices into one. It is defined by three strategies: *max*, *average*, and *weighted*. The selection filters possible matches from the aggregated matrix according to a defined strategy, e. g. see Fig. 7.2 (ii) and (iii). Possible strategies are *threshold*, *maxN*, and *maxDelta*. The direction is dedicated to a ranking of matched elements according to their similarity.

Matchers which use other matchers need to combine similarity values, e. g. the children and sibling matchers. This is covered by providing two strategies: *average* and *dice*. For more details on the strategies please see Sect. B.2 in the appendix.

These combination strategies grant the possibility of configuring the matching results by using the strategies and their parameters as outlined before. Typically, a default strategy is defined for a user of MatchBox. Following the matching process, having aggregated and selected the similarity values, the mappings are created.

## 7.3 Evaluation Data Sets

The goal of our data set selection was to have heterogeneous real-world and academic data that covers a variety of structural and linguistic properties. We selected two evaluation data sets, the ATL-zoo[1] as proposed by us in [151] and web service message mappings in an enterprise environment (ESR). To introduce both sets we will first define the metrics used to describe them, the data set statistics can also be found in [149]. We separated them because the first is an academic data set and the second is from an enterprise context of SAP. In addition, the ATL-zoo data sets consist of EMF [109] metamodels, where the ESR data is imported from schemas extracted from web service message mappings.

### 7.3.1 Data set metrics

The real-world and academic data sets we introduced are diverse in size, structure, naming, etc. and incorporate in sum 51 mappings. Since the metamodel definition or graph itself can be difficult to comprehend when exceeding a certain size we characterize them using metrics. We divide the metrics into the following three groups:

- Dimensions of metamodels and mappings,
- Linguistic properties and,
- Structural properties.

**Dimensions** The first metric is the size $s_m$ of a metamodel $m$ as defined in (7.1). It is defined as the number of elements $E_m$ of the metamodel $m$. Please note that the size does not contain the number of relations or labels.

$$s_m = |E_m| \tag{7.1}$$

The size of a metamodel is not sufficient information to characterize a mapping between two metamodels $s_s$ and $s_t$. Therefore, we also define the source-target element ratio as a metric to show the ratio of the size of metamodels mapped. We define the ratio (7.2) as the minimum of source and target metamodel size $s_s/s_t$ divided by the maximum of the source and target size.

$$r = \frac{Min(s_s, s_t)}{Max(s_s, s_t)} \tag{7.2}$$

Following the ratio, a mapping's size is also of interest, because it shows the number of matches which are to be discovered by a matching system.

[1]`http://www.eclipse.org/m2m/atl/atlTransformations/`

We define a mapping's size (7.3) as the number of matches $M$, which are a subset of the cartesian product of source $E_s$ and target elements $E_t$.

$$m_{\{s,t\}} = |M \supseteq E_s \times E_t| \tag{7.3}$$

Having defined a mapping's size and the source-target metamodel ratio, the next step is to investigate the coverage of a mapping. That is the average of the fractions of the mapping and metamodel sizes. That means how many metamodel elements are mapped relative to all elements.

$$c_m = \frac{1}{2} \cdot (\frac{m_s}{s_s} + \frac{m_t}{s_t}) \tag{7.4}$$

**Linguistic properties** In accordance with the matching techniques classification of Sect. 2.1.2.2, the linguistic properties can be given by the number of labels available. In the context of metamodels these lables are names.

Therefore, the name similarity and thus amount of linguistic information available for matching can be measured by the name overlap of the elements of the metamodels to be compared. The name overlap, as also used by [64], describes the ratio between the identical and all names. As defined in (7.5) it is the fraction of identical (overlapping) source element name labels $L_s^n$ ($n$ denotes the name) and target element name labels $L_t^n$ ($L_s^n \cap L_t^n$) and all name labels given by both metamodels ($L_s^n \cup L_t^n$).

$$N = \frac{L_s^n \cap L_t^n}{L_s^n \cup L_t^n} \tag{7.5}$$

Since the name overlap only describes a strict linguistic similarity by requesting identical names we relax the condition by using token overlap instead of names. We define a set of tokens as trigrams, e. g. as in the matching systems [100, 23] of a name tri(L), that is all substrings of size three. We took trigrams because it is a common approach for name matching and we define the metric as given in (7.6). The equation is similar to Eq. 7.5 but uses tokens rather than names.

$$T = \frac{tri(L_s^n) \cap tri(L_t^n)}{tri(L_s^n) \cup tri(L_t^n)} \tag{7.6}$$

**Structural properties** The structural properties are defined by the underlying graph structure, which we aim to utilize for matching. Since we make use of our $k$-max degree approach, we give values for the maximal degree of a metamodel's element, that is the number of neighbours or edges (except the root package element). For the definition please see Sect. 2.2, Eq. 5.5.

We define the edge count as another metric. The number of edges is defined as the number of relations $R_m$ of a metamodel $m$ as given in (7.7).

$$e_m = |R_m| \tag{7.7}$$

To measure the amount of information contained in a tree we define the tree-original ratio. The ratio captures the number of edges in a metamodel's tree $R_{tree}$ relative to the original graph $R_{graph}$. It is defined as in (7.8), which defines the fraction between the tree's edges and the graph's edges.

$$\delta_{tree} = \frac{|R_{tree}|}{|R_{graph}|} \tag{7.8}$$

Similar to the tree ratio we also define the planar-original graph ratio. That is the fraction of edges contained in the planarisation of a graph $R_{planar}$, and the edges of the original graph $R_{graph}$(see (7.9)).

$$\delta_{planar} = \frac{|R_{planar}|}{|R_{graph}|} \tag{7.9}$$

In the subsequent sections we will apply our metrics on the ESR and ATL-zoo data to characterise the mapping data sets used for our evaluation.

### 7.3.2 Enterprise service repository mappings

The enterprise service repository mappings are a heterogeneous real-world data set covering various business domains. It consists of more than 100 mappings[2]. These mappings are extracted from the ESR[3] of SAP. Moreover, they are heterogeneous in naming[4] as well as in structure.

The mappings have been manually exported from the ESR. Since they are defined in a SAP proprietary mapping format [83], they have been imported using a parser generated by a grammar specification using EMFText [57]. The grammar is given in Appendix A.2. The ESR mappings and metamodels are then automatically transformed into our internal mapping model using Java.

From the available ESR data we selected 31 mappings and consequently 62 participating metamodels. We based the selection on the matchability of the cases[5]. Please note that we did not remove existing duplicate schemas

[2]The domains included in this diverse data set are: financial, human resources, point of sale, health care, device integration, accounting, supplier relationship management (SRM), supply chain management (SCM), etc

[3]A central repository in which service interfaces and enterprise services are modeled and their metadata is stored[83].

[4]The languages used are English, German, and Spanish and there are metamodels that only use codes as naming, etc.

[5]The selection was based on a threshold of 0.2 for the F-Measure obtained by an average matching configuration. That means at least 20% of all mappings should be found by the standard MatchBox and be correct.

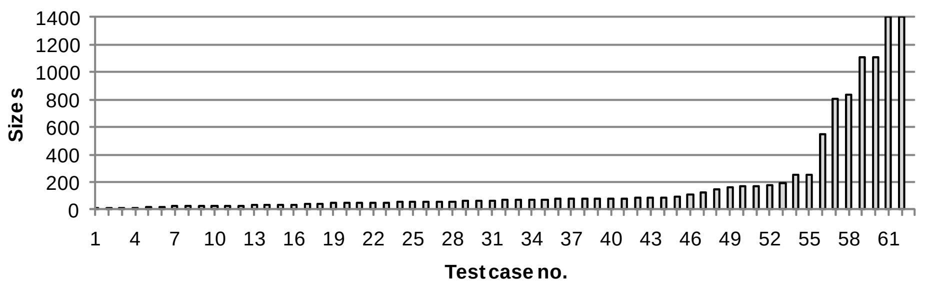


Figure 7.3: Size of ESR metamodels

since they participate in different mappings and are, therefore, also multiple times part of the matching.

**Metamodel size** The ESR mappings consist of the 62 metamodels participating in 31 mappings. The maximum size is 1,401 elements, where the smallest metamodel consists of 6 elements. Figure 7.3 depicts the sizes of all 62 metamodels sorted in an ascending order. It can be seen that most of them are below a size of 200 elements. However, we also identified 15 large-scale examples which are used to evaluate the scalability of our approach.

**Source target ratio** We depict the distribution of the source target element ratio $r$ in Fig. 7.4. To better illustrate the smaller examples we removed the ratio for examples with more than 500 elements (those are almost equal in size and thus close to the line). The centred line shows a one to one ratio that means source and target size are equal. Most of the mappings show a similar size ratio. However, there are extreme values which show a ratio of 17 to 479 being mapped onto each other. This raises the question of the mapping size and coverage, which will be answered in the following.

**Mapping size** The ESR mapping data set comprises 31 mappings with an average size of 150.09. The maximal size is 1,401 where the minimum is 4. The distribution of the mappings is given in Fig. 7.5. Most of the mappings are below 200 elements and thus not of large-scale dimension. However, eight mappings exceed the size of 500 and are not depicted for a better overview. The ratio of those mappings is almost one.

**Mapping coverage** The coverage of the ESR mappings, i. e. the ratio of source and target elements mapped, is depicted in Fig. 7.6. On average the coverage is 0.83, that means that on average 83% of source and target elements are mapped onto each other. The maximum is 1, which means all elements are mapped, where the minimum is 0.30. There only 30% of

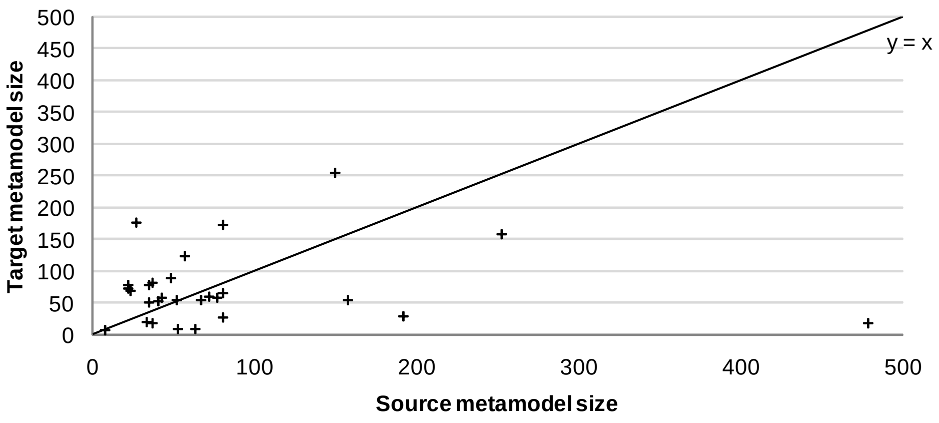


Figure 7.4: Source and target metamodel element ratio for ESR mappings with metamodels smaller than 500 elements

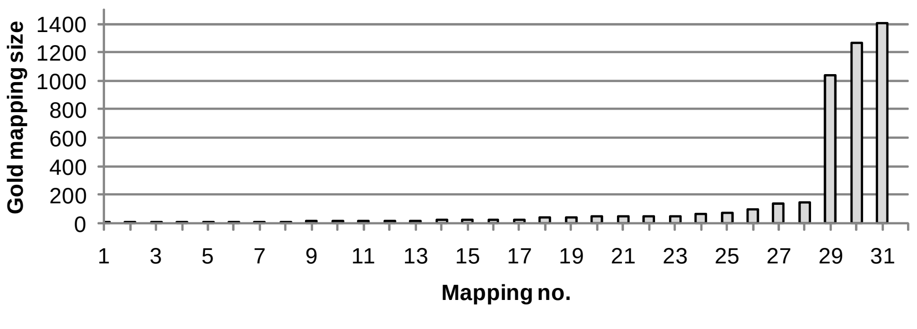


Figure 7.5: Size of ESR gold-standard mappings

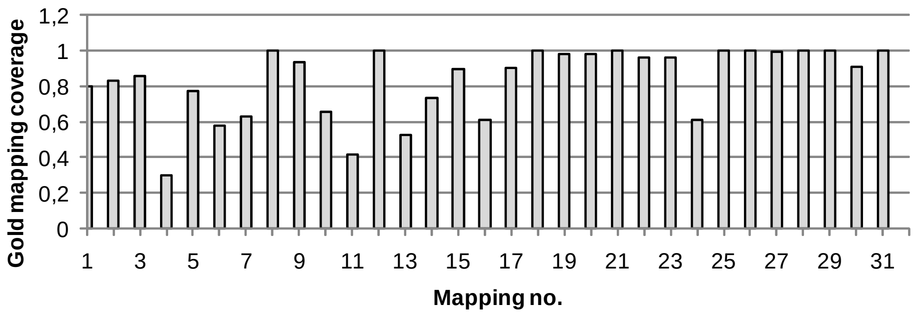


Figure 7.6: Coverage of ESR gold-standard mappings

all elements are mapped onto each other. Again these values underline the real-world properties of the ESR mappings and its considerable range of mappings.

### 7.3.2.1 Linguistic properties

The linguistic properties of the ESR mappings are investigated by the name overlap, and token overlap.

**Name overlap** The diagram in Fig. 7.7 shows the distribution of the name overlap of our ESR data (grey bars). 16 of all metamodel pairs show no name overlap, 4 are identical in their names with an overlap of 1.0, where 4 show only an overlap of 0.1. The remainder distributes between 0.1 to 1.0 overlap. This demonstrates that the ESR mappings include some identical mappings but a lot with different naming which indicates failure when applying name-based matching and thus improvements by structure-based approaches. This is also underlined by an average name overlap of 0.28.

**Token overlap** The token overlap is depicted in Fig. 7.7 (white bars). Complementary to the name overlap it depicts the number of overlapping trigrams. This time all mappings show a token overlap of at least 0.1. In addition, 17 cases have a token overlap of 0.2 or less which is also unsuitable for name-based matching but 6 cases have a token overlap of 1.0 which are the identical mappings and derivations thereof.

### 7.3.2.2 Structural properties

The structural properties for the enterprise service mappings are first measured by the number of edges available per metamodel, indicating the information available for matching. Figure 7.8 shows the number of edges for all ESR mapping metamodels ordered according to the metamodel's size. On

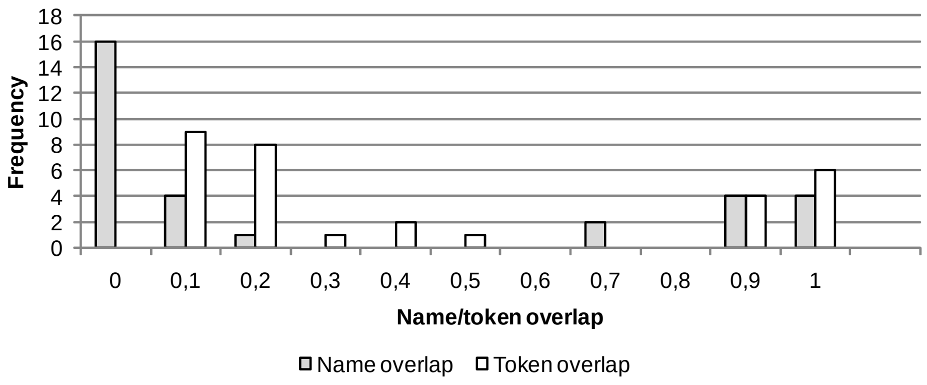


Figure 7.7: Name and token overlap for ESR source and target metamodels

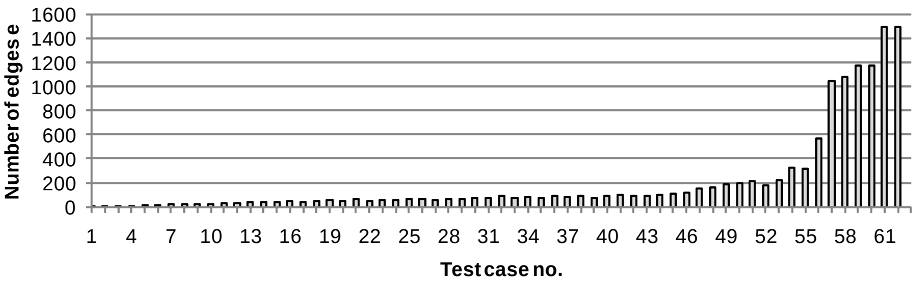


Figure 7.8: Number of edges for ESR metamodels

average a metamodel has 203.79 edges, where at most 1,491 edges exist, the minimum is 5 edges.

**Maximal degree** Since we follow a degree-based approach whenever a representative or element is selected we took values for the maximum degree of the ESR metamodels. We ordered them according to a metamodel's size in Fig. 7.9. The maximal degree has no direct connection to a metamodel's size, small metamodels do show a smaller max degree than large ones. The average maximal degree is 25.16, with a maximum of 141 and a minimum of 5. The average degree for all ESR metamodel elements including attributes is 1.13.

The argumentation of our approach is based on the amount of structural information and the superiority of planar graphs compared to trees. Therefore, we investigated the edge ratio of the ESR metamodels graphs w.r.t. a tree as well as to a planar graph representation.

**Tree-original graph ratio** The percentage of loss in edges comparing a tree or planar graph to the orginal graph is depicted in Fig. 7.10. The test

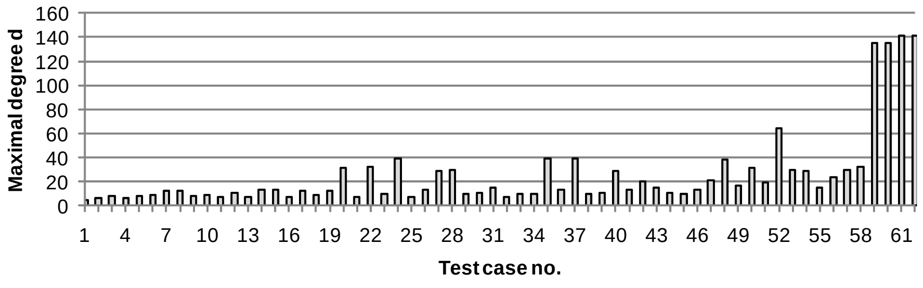


Figure 7.9: Maximal degree for ESR metamodels

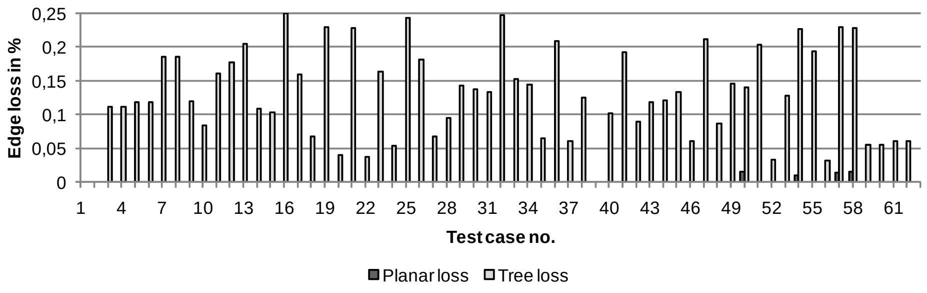


Figure 7.10: Loss in edges ($1 - \delta_{planar}$, $1 - \delta_{tree}$ respectively) for ESR metamodel graphs

cases are ordered in an ascending order of the metamodels size. The values of the tree edge loss show that independent of a metamodel's size only 3 metamodels are indeed trees, since they have zero loss. However, 59 of 62, i. e. 95% of all models are not trees. The loss of information is up to 0.25. On average 0.13 is lost, which means on average a tree of an ESR metamodel only contains 87% of structural information in terms of edges compared to the original metamodel graph.

**Planar-original graph ratio** In comparison to the tree original ratio, we also took the planar-original graph ratio. The value is on average 0.999% meaning almost all ESR metamodels are planar and can be used with our algorithms without any loss in structural information. The maximum loss of edges is 0.026%.

## 7.3.3 ATL-zoo mappings

The ATL-zoo mappings are a data set extracted from a publicly available source[6]. It is a collection of 109 ATL transformations serving educational and

[6]`http://www.eclipse.org/m2m/atl/atlTransformations/`

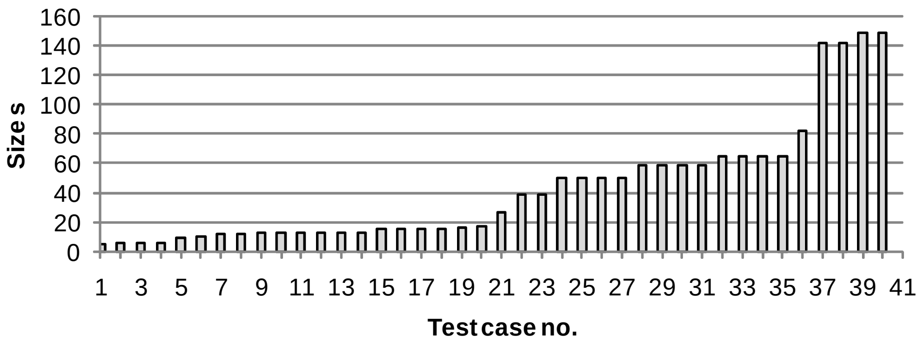


Figure 7.11: Size of ATL metamodels

academic purposes. These transformations are defined between two metamodels using the Atlas Transformation Language (ATL) [70] which defines a grammar for specifying mappings. These mappings express one-to-one as well as one-to-many correspondences. We used these transformations for matching evaluation by transforming them into our internal mapping model. A detailed description of our implementation can be found in Appendix A.1.

We selected 10 out of the 109 transformations based on their real-world closeness omitting educational and toy examples. We also omitted transformations whose metamodels are not based on EMF, because our matching framework MatchBox relies on it. Additionally, we took 10 examples from an academic investigation by Kappel et al. [73]. They used examples for integrating Ecore, UML, and WebML, which we easily imported into our internal data model, since their mapping model is close to ours. In the following we will detail the ATL data set similar to the ESR data set.

**Metamodel size** The size of the participating metamodels range from 5 to 142 elements with an average of 42.55 elements. The corresponding sizes are also depicted in Fig. 7.11 ordering them in ascending order. The metamodels are considerably smaller than the ones from the ESR data set which can also be seen in the size of the gold mappings.

**Source target ratio** The ratio between the source and target metamodel's sizes for the ATL-zoo metamodels is given in Fig. 7.12. Compared to the ESR mappings it shows a similar behaviour with most of the mappings being of rather small and similar size. The average source target ratio is similar to the ESR's ratio. Thereby, the values range from 0.13 to 1.00 with the biggest size mismatch being of 130 to 24 elements.

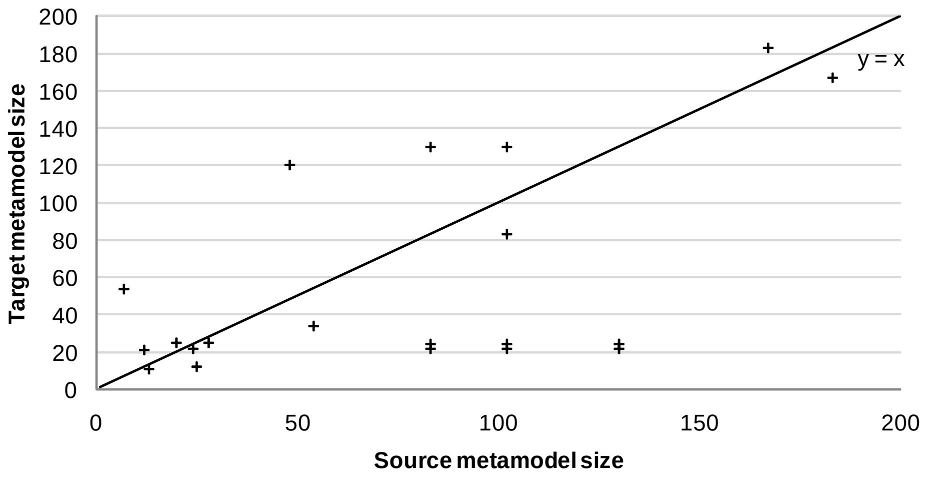


Figure 7.12: Source and target metamodel element ratio for ATL mappings

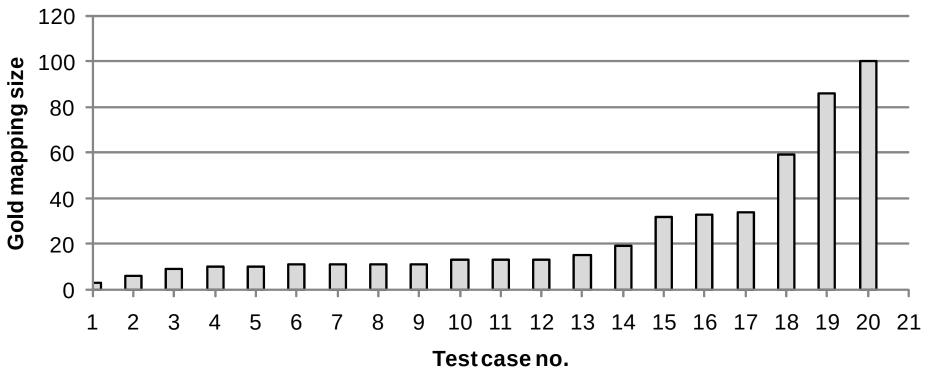


Figure 7.13: Size of ATL gold-standard mappings

**Mapping size** The mappings extracted from the ATL-zoo are at minimum of size 3 and at maximum 100. Furthermore, most of them are around the size of 10 elements as shown in the diagram of Fig. 7.13. The average size is 24.95 elements mapped for a source and target metamodel.

**Mapping coverage** The coverage of the mappings follows a lower dimension compared to the ESR, i. e. on average it is 0.35 for ATL in contrast to 0.85 for ESR. That means that the mappings cover fewer elements than the real-world mappings, which underlines the academic nature of the ATL-zoo mappings. The distribution of the coverage is shown in Fig. 7.14 ordered according to the gold mapping size.

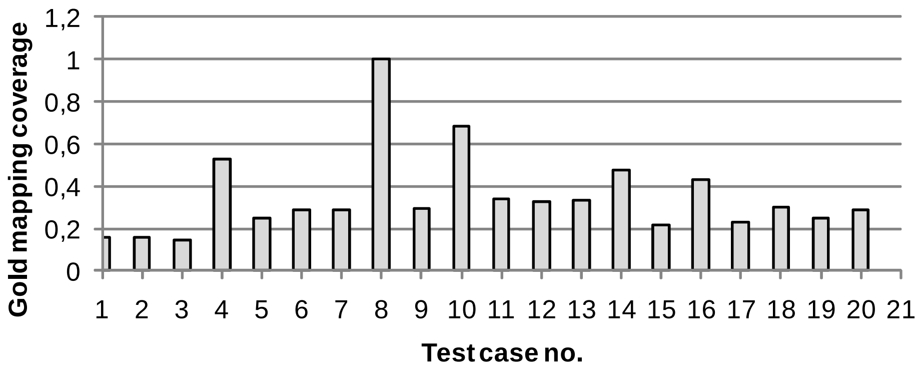


Figure 7.14: Coverage of ATL gold-standard mappings

#### 7.3.3.1 Linguistic properties

The linguistic properties of the ATL data set are again given by the overlap of names and tokens for each pair of metamodels participating in a gold-standard mapping (see Fig. 7.15).

**Name overlap** The ratio of identical names, that is the name overlap, is on average 0.17, with 9 showing a minimum of 0 and 2 a maximum of 0.74. The remaining metamodel pairs show values between 0.1 and 0.4.

**Token overlap** The token overlap is in a similar range, with an average of 0.24 and a minimum of 0 and maximum of 0.9, where only 1 of all 20 transformations has no overlap at all. The majority has an overlap of 0.1, one case shows an overlap of 0.85 (part of the class 0.9 in Fig. 7.15). This shows that the ATL-zoo has slightly lower linguistic information available compared to the ESR data.

#### 7.3.3.2 Structural properties

The structural properties are depicted in Fig. 7.16 in terms of edges ordered according to the corresponding metamodel's size in ascending order. Naturally, the number of edges increases with an increasing size in metamodels. On average a metamodel has 54.65 edges, with at most 177 edges and a minimum of 6 edges.

**Maximal degree** The maximal degree distribution for the ATL-zoo is shown in Fig. 7.17. The average maximal degree is 7.2 ranging from 3 up to 17. This is less than the 25.16 of the ESR mappings but can be reasoned by the metamodels' sizes. The average element degree including attributes is 1.33.

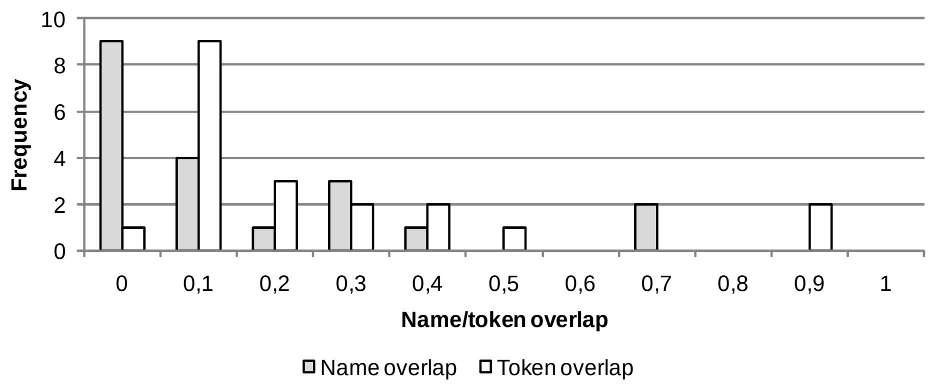


Figure 7.15: Name and token overlap for ATL source and target metamodels

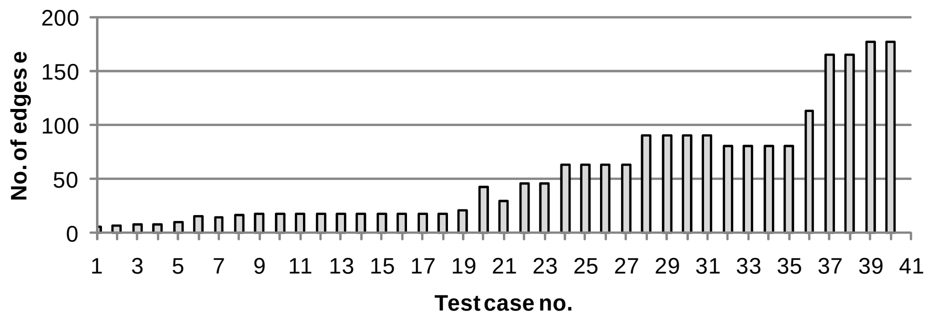


Figure 7.16: Number of edges for ATL metamodels

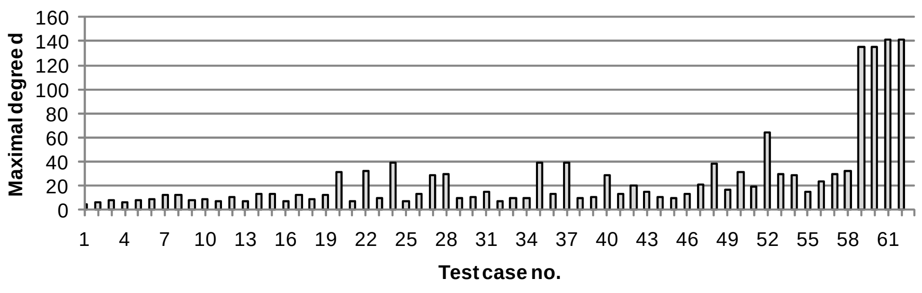


Figure 7.17: Maximal degree for ATL metamodels

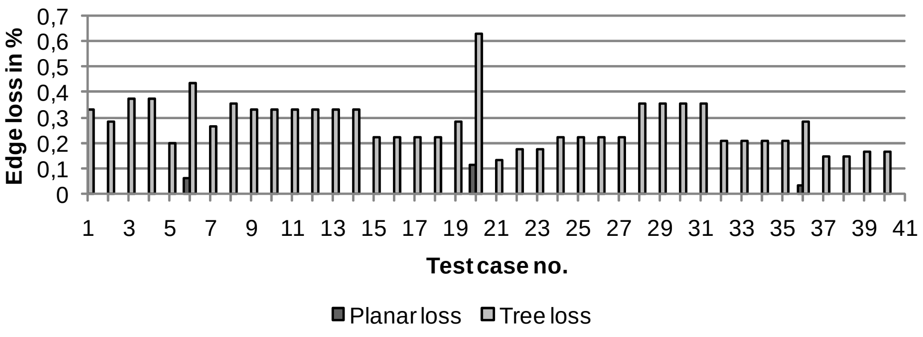


Figure 7.18: Loss in edges ($1 - \delta_{planar}$, $1 - \delta_{tree}$ respectively) for ATL metamodel graphs

**Tree-original graph ratio** The ratio of edges between a tree and a metamodel's graph is depicted in Fig. 7.18. None of the ATL-zoo metamodels is a tree. The maximal ratio is 0.87. This also shows that at least 13% of all edges are lost and not available for matching. Furthermore, the minimum ratio is 0.37 which implies a considerable loss of 63% of edges. On average more than one quarter of all edges are not part of a tree given by an average ratio of 0.72.

**Planar-original graph ratio** In contrast to trees, a planar graph holds more information as given in the planar-original graph ratio in Fig. 7.18. Only 3 of the 40 metamodels are not planar which means that for 37 metamodels no information is lost. The remaining three have a loss in ratio of 4%, 7%, and 12% respectively. That means that almost all edges are preserved by making the graph planar. The amount of information contained is consequently more than in a corresponding tree, since the planar-original graph ratio has an average ratio of 0.99%.

### 7.3.4 Summary

Both data sets ESR and ATL are used by us for evaluating our matching system. The ESR data set shows a wider range of mappings and also in terms of size provides large-scale examples. In contrast, the ATL Zoo metamodels show more structural information than the ones from ESR, which is due to the fact that the ESR metamodels are extracted from schema definitions which serve as a basis of exchange in the SAP PI system [83]. Schemas by their nature have less structural information, because they are built around tree definitions which become graphs by type references. These schemas are used as metamodels on a type-based graph. That means the type information is preserved in a graph structure rather than flattened and converted to

| | ESR | | | ATL | | |
|---|---|---|---|---|---|---|
| Metric | Avg. | Max. | Min. | Avg. | Max. | Min. |
| Metamodel size | 180.94 | 1,401.00 | 6.00 | 42.55 | 142.00 | 5.00 |
| Source target ratio | 0.55 | 1.00 | 0.04 | 0.87 | 1.00 | 0.19 |
| Mapping size | 153.94 | 1,401.00 | 4.00 | 24.95 | 100 | 3.00 |
| Mapping coverage | 0.83 | 1.00 | 0.30 | 0.35 | 1.00 | 0.15 |
| Name overlap | 0.28 | 1.00 | 0.00 | 0.17 | 0.74 | 0.01 |
| Token overlap | 0.40 | 1.00 | 0.02 | 0.24 | 0.85 | 0.03 |
| Maximal degree | 25.16 | 141.00 | 5.00 | 7.20 | 17.00 | 3.00 |
| Edge size | 203.79 | 1,491.00 | 5.00 | 54.65 | 177.00 | 6.00 |
| Tree ratio | 0.87 | 1.00 | 0.75 | 0.72 | 0.87 | 0.37 |
| Planar ratio | 1.00 | 1.00 | 0.98 | 0.99 | 1.00 | 0.88 |

Table 7.1: Comparison of metrics for ESR and ATL data set

a tree. Table 7.1 shows an overview of the numbers presented to characterize and compare the ATL and ESR data sets.

We conclude that the ESR and ATL data sets provide heterogeneous examples in size, linguistic, and structural properties. Consequently, they provide a profound basis for our evaluation as given in the following.

## 7.4 Evaluation Criteria

To evaluate the matching quality we use the established measures: precision $p$, recall $r$, and F-Measure $F$, which are defined in [129] as follows. Let $t_p$ be the true positives, i. e. correct results found, $f_p$ the false positives, i. e. the found but incorrect results, and $f_n$ the false negatives, i. e. not found but correct results. Then the formulas for these measures are as in (7.10):

$$\begin{aligned} p &= \frac{t_p}{t_p + f_p} \\ r &= \frac{t_p}{t_p + f_n} \\ F &= 2 \cdot \frac{p \cdot r}{p + r} \end{aligned} \tag{7.10}$$

- *Precision* $p$ is the share of correct results relative to all results obtained by matching. One can say precision denotes the correctness, for instance a precision of 0.8 means that 8 of 10 matches are correct.

- *Recall* $r$ is the share of correctly found results relative to the number of all results. It denotes the completeness, i. e. a recall of 1 specifies that all mappings were found, however, there is no statement about how many more (incorrect) matches were found.

- *F-Measure F* represents the balance between precision and recall. It is commonly used in the field of information retrieval and applied by many matching approach evaluations, e. g. [98, 72, 37, 38, 151].

The F-Measure can be seen as the effectiveness of the matching balancing precision and recall equally. For instance, a precision and recall of 0.5 leads to an F-Measure of 0.5 stating that half of all correct results were found and half of all results found are correct.

It is important to note that, when we refer to average precision, recall, and F-Measure we took the average of those measures separately. That means an average F-Measure is the average of all F-Measures and not calculated as the F-Measure from the average precision and recall.

## 7.5 Results for Graph-based Matching

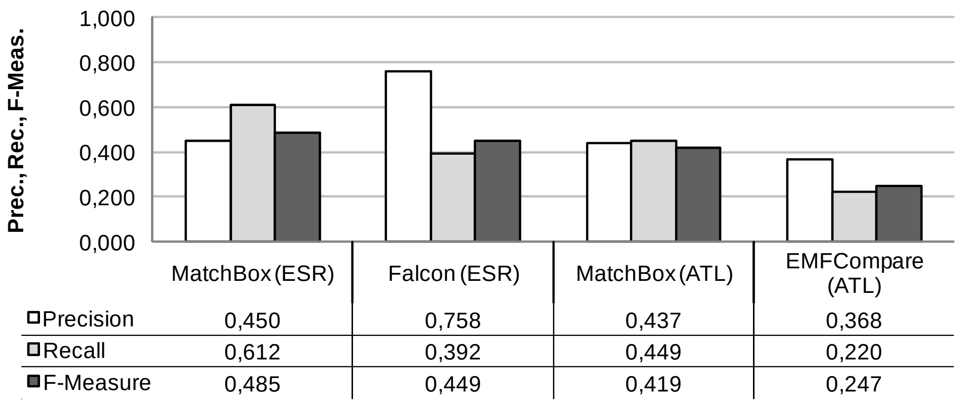
The answer to the question: "To which degree does our planar graph edit distance matcher improve result quality?" will be given in this section. Thereby, we followed the approach to first take values for the best average combination using our original system without our graph matchers. We then exchange one of the original matchers for our graph matchers and measure the quality again. The resulting delta shows the expected improvements of our approach. The detailed series can be found in [149].

**Baseline** Precision, Recall, and F-Measure serve as a basis for comparison of the result quality of our GED matcher and our mining matchers with our baseline of established matching techniques implemented in the MatchBox system. In order to define a baseline representing the best configuration of MatchBox we first determined the optimal combination of matchers achieving the highest F-Measure. That means the best suited number and the choice of specific matchers along with a fixed threshold. We investigated combinations of 4 to 6 matchers[7] and varied the parameters: combination strategy, selection strategy, threshold, MaxN, and delta (cf. Sect. B.2 in Appendix B) in order to identify the configuration leading to the best results.

As our MatchBox system applies tree-based matching techniques[8] several tree definitions have been investigated by us in [151]. Possible trees are based on the containment, inheritance, or reference relations of a metamodel. We conclude with the containment hierarchy (see Sect. 2.2.2 for different representations) performing with the best F-Measure [151]. The corresponding configuration consists of the four matchers name, parent,

[7]The range of 4 to 6 matchers has been confirmed by [23] as giving best results.

[8]State of the art matching techniques implemented operate on a tree, e. g. parent, children, sibling, and leaf matchers. Therefore, for comparison we need to investigate an optimal tree although it contains less information than a planar graph.



| | MatchBox (ESR) | Falcon (ESR) | MatchBox (ATL) | EMFCompare (ATL) |
|---|---|---|---|---|
| □ Precision | 0,450 | 0,758 | 0,437 | 0,368 |
| □ Recall | 0,612 | 0,392 | 0,449 | 0,220 |
| ■ F-Measure | 0,485 | 0,449 | 0,419 | 0,247 |

Figure 7.19: Baseline quality results for the ESR and ATL data sets

children and sibling matcher. For an optimal F-Measure these linguistic and similarity propagation-based matchers are configured with a fixed threshold of 0.3, a delta of 0.04 and average combination for single matcher results.

This configuration leads to the results as given in Fig. 7.19. The first group of bars on the left states the results for the ESR data set with precision 0.45, recall 0.612, F-Measure 0.485. To demonstrate the quality of our matching system and to verify that the data sets are hard we applied third-party matching systems. For the ESR data set we took the established ontology matching system Falcon [69], because it is also capable of matching schemas. The results for the Falcon system are an average precision of 0.758, a recall of 0.392, and F-Measure of 0.449. It can be seen that the Falcon system shows a higher precision than MatchBox but a lower F-Measure. This is due to the optimized configuration of Falcon for precise results (precision), while we optimized the configuration of MatchBox for F-Measure[9].

The second group from the right depicts the baseline for the ATL data set with values of precision 0.437, recall 0.449, and F-Measure 0.419. As third-party matching system we took the most prominent approach EMF Compare [10] instead of Falcon, because in contrast to EMF Compare, Falcon is not tailored to metamodels. The values for EMF Compare are depicted in Fig. 7.19 on the right. The values are considerably lower (MatchBox F-Measure 0.419 vs. 0.247 for EMF Compare ), because EMF Compare is made for model differencing. That means their assumption is a high similarity of the input, which is the case for differencing. However, they state that their task is model comparison, thus we took their approach as comparison. We

[9]We could provide a corresponding configuration with similar values for precision but a lower F-Measure, but this leads to the discussion of precision vs. recall. A high precision could be preferred over a high recall, i. e. more correct but less complete results. However, recall could also be preferred over precision because a user has to check all mappings anyway. Therefore, we optimized our evaluation to F-Measure to represent the harmonic mean between both measures.

would like to point out that EMF Compare also provides an extension mechanism which allows to integrate arbitrary matching techniques. Consequently, MatchBox or parts of it may be integrated in EMF Compare effectively increasing the result quality.

To conclude, it can be seen that the F-Measure values of third-party systems are below ours, demonstrating effectively the hardness of our matching tasks and confirming that only about half of all matches are found and correct for our real-world data sets.

### 7.5.1 Graph edit distance results

Our evaluation of our Graph Edit Distance matcher (GED) consists of the following three incremental steps:

1. We start with a comparison of the graph edit distance matcher added to MatchBox using reference-based graphs (Sect. 2.2.2) for the ESR and ATL data sets.
2. Then we show further improvement by considering the inheritance graph (Sect. 2.2.2) using our GED matcher in case of the ATL data set. Thereby, the results are combined with the ones obtained by the reference-based GED matcher.
3. Subsequently, we present our $k$-max degree approach showing its benefits with increasing $k$ for the ATL data set.

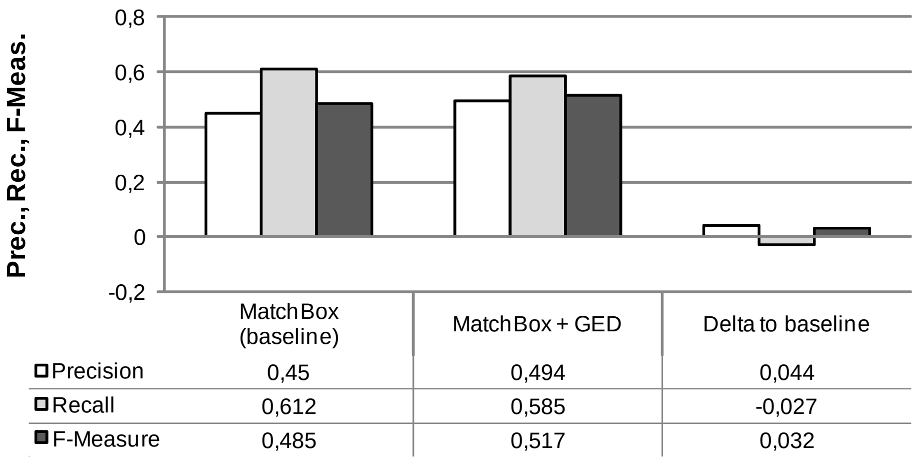
The subsequent sections will discuss and present each of the aforementioned steps.

#### 7.5.1.1 Reference-based GED for ESR

We first compared the graphs based on reference representations, i. e. inheritance is not considered. Since the ESR data set is made of business schemas it contains no inheritance, thus we do not need to differentiate between the two graph modes possible. For this purpose, we exchanged the leaf matcher with our planar GED matcher, because the leaf matcher was contributing less to the overall result than the other matchers [150]. The results by adding our reference-based graph edit distance matcher are depicted in Fig. 7.20 in the centre.

The numbers of precision 0.494, recall 0.585, and F-Measure 0.517 (in the centre) show an improvement compared to the baseline depicted on the left of the Figure. The delta obtained by our GED approach shows an increase of 0.044 (4.4%) in precision, a small loss in recall of 0.027 (2.7%) which yields an increase of 0.032 in F-Measure (3.2%).

For a detailed discussion of this distribution please see Sect. 7.7. In this section we show that our approach in the best case improves the precision



| | MatchBox (baseline) | MatchBox + GED | Delta to baseline |
|---|---|---|---|
| □ Precision | 0,45 | 0,494 | 0,044 |
| □ Recall | 0,612 | 0,585 | -0,027 |
| ■ F-Measure | 0,485 | 0,517 | 0,032 |

Figure 7.20: Result quality and delta for the ESR data set using the baseline MatchBox configuration and MatchBox with the GED matcher

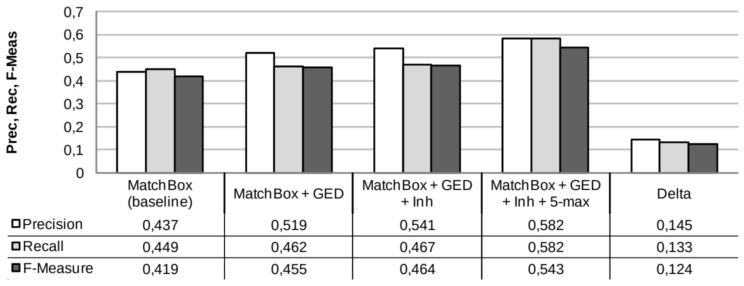
significantly by 0.454 (45.4%), where at worst it decreases the precision by 0.046 (4.6%). The recall only changes slightly which is a clear indication for the precision improvement by the GED.

#### 7.5.1.2 Reference-based and inheritance-based GED for ATL

The ATL data set of the MDA community consists of metamodels which also make use of inheritance, e. g. the UML metamodel in an extensive manner. Therefore, we investigated different graph representation modes as introduced in Sect. 2.2.2. The best mode w.r.t. our evaluation is the representation based on no inheritance, i. e. reference edges are not mixed with inheritance edges. Combining MatchBox and a GED matcher based on the reference graph without inheritance matching, the complete ATL test set yields a precision of 0.519, recall 0.462, and F-Measure 0.455. Compared to the original MatchBox, this is an improvement of 0.082 (8.2 %) in precision, 0.013 (1.3 %) in recall, and 0.036 (3.6 %) in F-Measure.

Since the inheritance information would be discarded, we added a second planar GED matcher (Inh) based on the inheritance edge graph. It is then combined with the results of the other matchers. The final numbers of this matcher using inheritance as in Fig. 7.21 (MatchBox + GED + Inh) are: precision 0.541, recall 0.467 and F-Measure 0.464. This leads to an improvement in F-Measure of 0.9 % w.r.t. the planar GED and of 5.8 % compared to the original MatchBox system. The main improvements are in precision (10.4 %), i. e. the mappings presented are more likely to be correct.

Finally, we took the values for both GED matchers and a 5-max degree approach, where we chose 5 based on a conservative estimation of low ef-



| | MatchBox (baseline) | MatchBox + GED | MatchBox + GED + Inh | MatchBox + GED + Inh + 5-max | Delta |
|---|---|---|---|---|---|
| □ Precision | 0,437 | 0,519 | 0,541 | 0,582 | 0,145 |
| ▨ Recall | 0,449 | 0,462 | 0,467 | 0,582 | 0,133 |
| ■ F-Measure | 0,419 | 0,455 | 0,464 | 0,543 | 0,124 |

Figure 7.21: Results quality for ATL data set using the GED matcher and corresponding delta

fort for a user, but still achieving improvements w.r.t. the original system's baseline. Figure 7.21 depicts the final numbers of: 0.582 for recall, 0.582 for precision and 0.543 for F-Measure[10]. We achieved an improvement compared to the baseline of 12.4% increasing the F-Measure from 0.41 to 0.54, i. e. a relative increase of 31.7%. In the following we will describe the additional improvements using our $k$-max degree approach.

#### 7.5.1.3 $k$-max degree GED for ATL

The evaluation of our $k$-max degree approach is based on the same set of ATL-zoo test data. We applied only our GED reference matcher and GED inheritance matcher to measure the improvement of the calculation. We assume a one-time input by the user by giving correct matches ($Seed(k)$) for the given problem of $k$ vertices. We took values for $k$ from 0 to 20, because for greater k, the more values remain unchanged. The $k$=0 denotes the reference value, i. e. matching without seed mappings. The user-input was simulated by querying the gold-standards for corresponding mappings.

As discussed at the end of this chapter in Sect. 7.5.1.4 we refrained from taking the values for the ESR data set. The short reason is the nature of the ESR data set which would require the GED matcher to start multiple calculations at multiple elements, because the elements and thus seeds are less connected than in ATL. This feature is currently not supported by our implementation, since it raises research questions for result merging.

We separate the results into two series using the ATL data set: (1) use $k$-max and (2) use and add $k$-max. The first series should demonstrate the impact of seed matches on the similarity calculation. Therefore, we did not add the seed input matches after calculation to avoid influencing the final results by introducing correct additional matches. The second series preserves

[10]Please note that the F-Measure is not based on a F-Measure calculation of both averages in precision and recall. Instead, it is the average of the F-Measures taken.

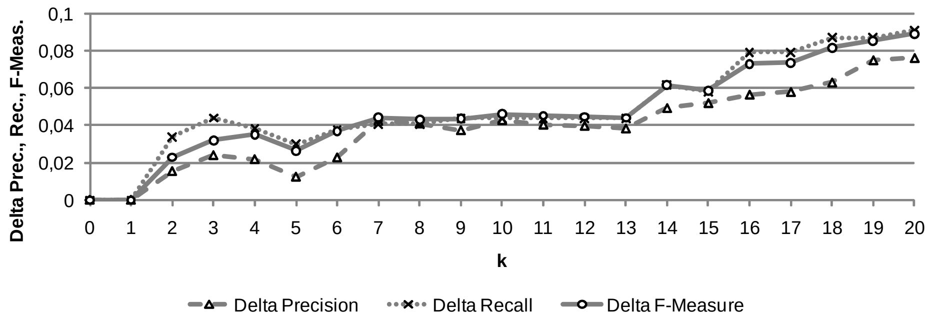


Figure 7.22: Delta of precision, recall, and F-measure for increasing k input mappings used only by the GED matcher for the ATL data set

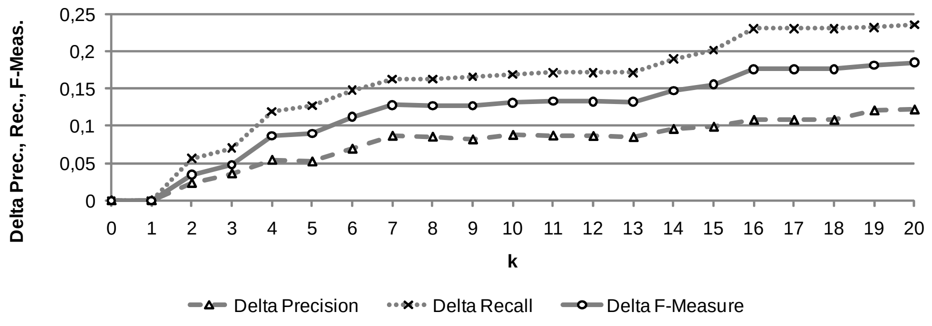


Figure 7.23: Delta of precision, recall, and F-measure for increasing k input mappings used by the GED matcher and added to final result for the ATL data set

the seed matches by adding them after the matching, presenting the results for a real-world usage, because a user is interested in all matches.

Figure 7.22 depicts scenario (1), i. e. using only the GED matcher with the given $k$-max correct matches as input for calculation, but not adding the correct matches to the overall result, in case of the ATL data. The delta between $k = 0$ and $20$ is shown to demonstrate the benefit of the GED matcher. The measurements are increasing, however they decrease at $k = 5$, because the structural vertices provide less correct additional information than for $k = 4$.

Figure 7.23 depicts the results for all three quality metrics when the input matches are added to the final results. Increasing $k$ improves the results and for $k = 3, 5, 8, 17$ the slope is maximal which is due to the nature of our test data. Overall, we achieve an increase in quality by increasing $k$. We can conclude that the minimal $k$ should be three, since our evaluation shows considerable increase from this on. We also demonstrated, that our GED is capable of discovering new matches by using seed matches.

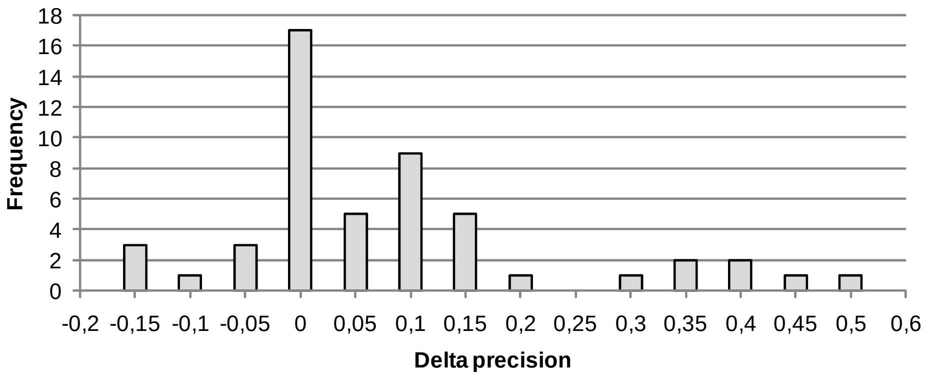


Figure 7.24: Distribution of precision delta comparing the MatchBox baseline to MatchBox with the GED matcher for ATL and ESR data sets

#### 7.5.1.4 Summary

Our evaluation based on 51 gold-standards (ESR and ATL) has investigated the quality of the planar GED comparing it to the baseline of the original MatchBox. Combining the numbers from the ESR and ATL data we noted an improvement of 0.07 (7 %) in precision and decrease of 0.009 (0.9 %) in recall leading to an overall improvement of 0.04 (4 %) in F-Measure.

Figure 7.24 shows the distribution of the delta of precision for ATL and ESR for a detailed explanation of the GED behaviour. It shows the deltas and their frequency, which illustrates the increase and decrease in precision of our approach. We selected precision, because F-Measure shows a similar behaviour but not as expressive as precision, while the recall remains almost unchanged.

Indeed there are 7 cases in which the precision shows a loss of up to 0.15 (15 %). However, the loss can be explained by two factors: (1) a metamodel size mismatch, and (2) a low token overlap. A source-target metamodel ratio of 0.2 and lower produces a decrease in quality, because it leads to a high edit distance calculated and consequently to a low similarity. However, this behaviour coincides with a token overlap below 0.1, i. e. only 10% of all element name parts are similar. Since our GED relies partly on linguistic similarity the structure is insufficient to neglect the effect of the low token overlap, thus our GED calculates wrong matches and decreases the result quality. Both in combination, the size mismatches and token overlap, may serve as a basis for decision for whether the GED matcher should be applied or not, thus compensating for its disadvantages.

On the other hand, there are four cases with an increase of up to 0.454 (45.4% assigned to the 0.5 bin). This considerable number is again explained by the source-target ratio and token overlap. For each of the cases the source-target ratio exceeds 0.5 and coincides with a token overlap of

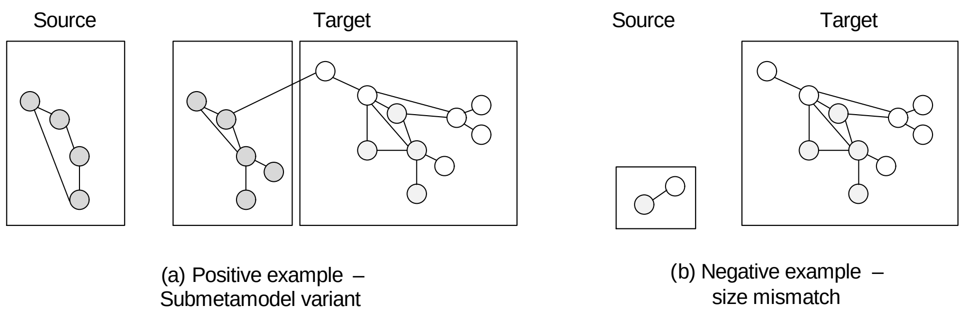

Source
Target
(a) Positive example – Submetamodel variant
Source
Target
(b) Negative example – size mismatch

Figure 7.25: Positive and negative example for quality changes by the GED matcher

at least 0.4. Consequently, this demonstrates that our GED matcher needs linguistic information and that in those cases the GED can be applied with result quality gains.

There exist three special cases with gains in precision, but with a size ratio or token overlap below the mentioned thresholds of 0.2 and 0.1 respectively. In these cases one metamodel is similar to parts of another from a structural point of view. The GED is effective in that case and can be applied for improved matching results.

Figure 7.25 depicts a positive and negative example illustrating the GED's behaviour. On the left hand side (a) one of the metamodels to be mapped is a variant of a submetamodel of the other, which is the case for the special improvements of 0.12 and 0.14. Misleading matches are eliminated, thus improving the precision considerably. The negative example in Fig. 7.25 (b) illustrates the size mismatch which results in a quality loss of up to 15% in precision.

An interesting question is posed by the following observation. Although we applied graph-based techniques which allow for up to 25% more information the results improved by only up to 15%. The reason is the overall matching process. The previous matching does not solely use a tree-based structure but also applies the name matcher. This matcher calculates its results without any knowledge about structure and similarities for the cartesian product of all elements. Consequently, almost all matches are found (high recall) but with a low precision. The traditional tree-based techniques refine the result, which our GED does as well, exceeding their results, but not in the aforementioned dimensions because the structure is still not sufficient to capture all semantics and thus mappings.

**Applicability** We deduce from our observations that the planar graph edit distance matcher can be applied to any metamodel, except those which show a source-target ratio below 0.2 (w.r.t. our data set) and a token overlap below 0.1. In addition, we observed that the GED produces good results for a source-target ratio above 0.5 and a token overlap above 0.4.

### 7.5.2 Graph mining results

The evaluation of both graph-mining based matchers is similar to the GED evaluation. We first derive optimal configurations of the mining matcher parameters to then compare their results to our baseline, i. e. the original MatchBox system.

Recapitulating, the mining-based matching is performed by pattern mapping and embeddings mapping. This processing order is also followed by us in our evaluation. That is, first we discuss parameters such as maximal pattern size to cope with the complexity of the pattern extraction and the pattern mapping. Then we discuss the relevance-based filtering of patterns to reduce the number of comparisons and embedding mappings. Finally, we evaluate the overall quality achieved by mapping of patterns. We will present both mining matchers in each phase in the following. We used the ATL and the ESR data sets to obtain our values.

#### 7.5.2.1 Preliminaries

As already mentioned, the cause problem of graph mining is its exponential complexity. Even though we applied the optimization of using closed graphs the problem still remains. To prevent it from restricting our implementation to a certain size several solution strategies can be applied:

- Process abortion after a pre-defined time out,
- Definition of a maximal pattern size,
- Filtering of patterns.

The most obvious limitation is the runtime itself. After mining for a specific time, the algorithm can be aborted. Already found patterns can be processed in the following phases. In tests, the time limit was set to 15 seconds and was reached only in the largest test cases (1,401 elements). We also noticed during our evaluation that graph mining applied to identical metamodels explodes both in number of patterns and runtime. This is due to the fact that two identical metamodels contain any combination of patterns in size and position. Therefore, we conclude that our mining-based matching should not be applied for identical metamodels but these can be easily detected by pre-processing (token overlap and source-target ratio). Consequently, we removed all identical mapping test cases for this part of the evaluation.

#### 7.5.2.2 Pattern mining and mapping

The parameters in question for the extraction phase are: a time out, a maximal pattern size, and a filtering of patterns. Since the time out is not needed for our data set, we evaluated the size restriction and the filtering.

Since the algorithms spend most of the calculation time on big patterns (more than 12 elements), we evaluated a maximum size restriction of patterns to mine. Our design pattern matcher and the redundancy matcher are both exponential in runtime with respect to the maximal size of found patterns, because both algorithms calculate subgraph isomorphisms for patterns. Additionally, the design pattern matcher also depends on the number of existing patterns because it searches for all of them in contrast to the redundancy matcher. This fact also causes exponential memory consumption dependent on pattern size for the design pattern matcher. Evaluating maximal pattern sizes from 5 to 30 we concluded that limiting the size to 12 performed best in terms of runtime, since the exponential growth led to a runtime of minutes for certain examples larger than 13[11].

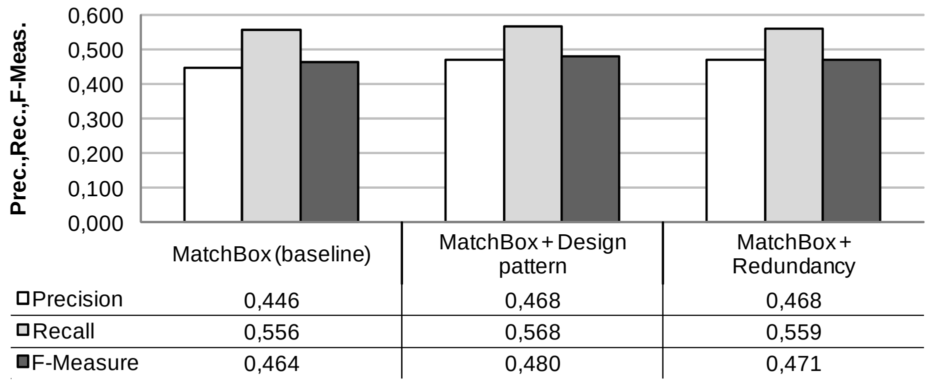
The relevance-based filtering of patterns uses two criteria to decide if a given pattern should be filtered: size and frequency. Therefore, we measured the combination of both relative to the F-Measure obtained. The result for a maximal average F-Measure was $\alpha = 2$ and $\beta = -1$ with a threshold for linguistic classes of $t = 0.5$ [112]. This means that the quadratic pattern size decides over a pattern frequency on the pattern's relevance. In other words, the bigger a pattern the more important it is, the more frequent, the less important.

Another possibility is to filter according to a pattern's type. For instance, trivial patterns which are attribute element relationships need to be filtered. The patterns that were found were categorized as given in the corresponding diploma thesis [112]. There, the patterns were separated into trivial patterns (attribute-class relations), star (multiple attributes-class), list (relations forming a path), trees (reference or inheritance tree), and graphs (any other pattern).

We observed that our design pattern matcher discovered in 19 of 46 mappings patterns, most of these patterns are from the ATL-zoo test data. In 13 cases those patterns were graphs and in 12 cases tree patterns. It also discovered in 18 cases list patterns, in 13 star patterns, and in 19 trivial patterns. However, in 19 cases patterns had been filtered out.

The redundancy matcher mined patterns in 43 out of 46 metamodel pairs (mappings). Thereby, in only two cases tree patterns were found. The redundancy matcher mined lists in 32 cases, and in 27 cases star patterns. This underlines the mining for redundant information which is local infor-

[11] We performed a detailed analysis of pattern sizes in the context of a diploma thesis in [112]



| | MatchBox (baseline) | MatchBox + Design pattern | MatchBox + Redundancy |
|---|---|---|---|
| □Precision | 0,446 | 0,468 | 0,468 |
| □Recall | 0,556 | 0,568 | 0,559 |
| ■F-Measure | 0,464 | 0,480 | 0,471 |

Figure 7.26: Result quality for design pattern, redundancy, and baseline matchers

mation not conforming to a design pattern, where in contrast the design pattern matcher mined more graph and tree patterns.

#### 7.5.2.3 Embedding mappings and results

Using the previous insights for the different parameters, the quality of the design pattern and redundancy matcher was compared to the baseline matching system. Figure 7.26 shows the results obtained by the matchers.

An overall improvement of the quality could be achieved by adding the design pattern matcher. Precision as well as recall could be improved meaning that incorrect mappings got filtered and new mappings could be found. The overall average precision improved by 2.2% while the recall for all mappings was improved by 1.2%. Thereby, the improvements were limited to 11 of the test cases, where for the others either no patterns were mined or the result remained unchanged. The results of the redundancy matcher are different. The ESR and ATL data sets both show redundant information and thus patterns which leads on average to the results as shown in Fig 7.26, i. e. 0.5% in precision and 0.3% in recall.

The evaluation showed that the design pattern matcher is best applicable in small- to mid-sized transformations. The redundancy matcher obtained best results in large models with much redundant and structural information. It could only improve these special case transformations and is not applicable in general as the average numbers for F-Measure show. The use of linguistic information during the mining process is essential for the precision of the mappings as well as for the runtime of the algorithms. Nonetheless additional limits for runtime and maximal pattern size are needed.

### 7.5.3 Discussion

The high complexity in runtime and memory of the mining algorithms used demanded for artificial limitations of the complexity, i. e. a maximal pattern size and a frequency-based pattern filtering. This results in a decrease of the patterns found and their applicability for matching. That means only patterns of limited size (12) were found, hence a divergence of the quality has to be accepted. Therefore, we conclude that mining-based matching can be applied, but on average it only shows little improvements. This opens directions for further research, for instance how to apply light-weight matching techniques to reduce the search space instead of using filtering.

**Applicability** The mining can be applied to any metamodel except identity mappings and metamodels without any patterns. Both properties are easily identifiable via pre-processing, still there exist metamodels with patterns which do not benefit from our approach.

## 7.6 Results for Graph-based Partitioning

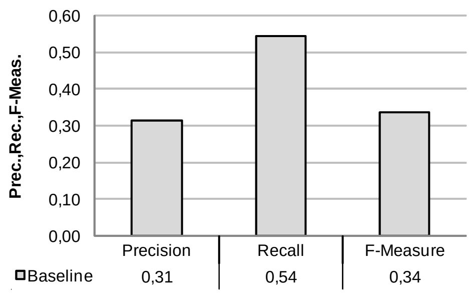
Our proposed planar partitioning based on the planar edge seperator (PES) and the corresponding four assignment algorithms should tackle the problem of large-scale metamodel matching (Chap. 6). Therefore, we identified the questions: Which partition size is the optimal choice w.r.t. to maximal result quality? To which degree does our planar partitioning reduce memory consumption? Which assignment algorithm should be used for runtime reduction while minimizing the loss in result quality? To answer these question we determined the best configuration for the following parameters: partition size, partition algorithm, and assignment algorithm (see [149] for the series). We apply the following incremental approach:

1. First, we apply partitioning only, i. e. planar partitioning and two comparable algorithms. We determine their quality, memory consumption as well as runtime and conclude with the choice of the PES.

2. Subsequently, we investigate the behaviour of the assignment algorithms to demonstrate the effectiveness of the generalized assignment.

3. Having selected a partitioning and assignment algorithm we finally present a comparison between partition-based matching and the baseline.

In the following we shortly describe the measurements used, to then give our results for the partition algorithms, and subsequently discuss the assignment algorithms.



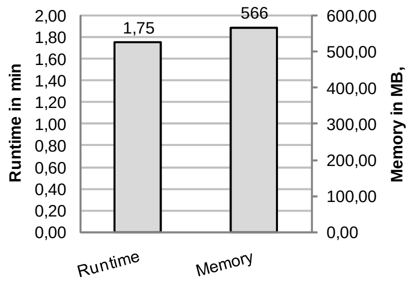


Figure 7.27: Quality and runtime baseline for 15 ESR large-scale mappings without partition-based matching

**Metrics** The result quality is characterized by the introduced metrics precision, recall, and F-measure. The scalability is given by using the runtime and memory consumption.

**Runtime** If not stated otherwise runtime refers to the complete matching process in seconds including partitioning, partition assignment, and partition matching. Otherwise, we will state explicitly which phases runtime was measured for.

**Memory consumption** Memory consumption is given by the average of memory in megabyte (MB) used during the complete matching process. We measured memory and runtime using the built-in facilities provided by Java, having a separate memory observing thread and averaging each number over 10 runs.

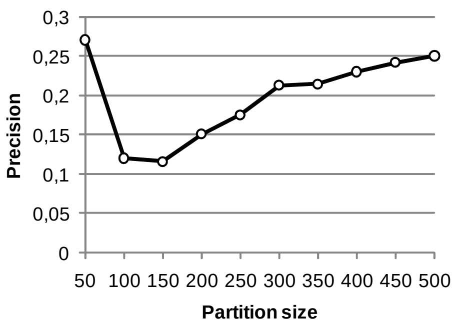
The resulting baseline numbers for our baseline system (without partitioning) are given in Fig. 7.27[12]. The numbers for precision, recall, and F-Measure are 0.31, 0.54, and 0.34. The corresponding memory consumption has a maximum of 566 MB with a runtime of 1.75 min.

## 7.6.1 Partition size

First, we need to identify the data set as candidate for evaluating our partitioning. Therefore, we compared MatchBox with and without our graph-based partitioning, respectively applying the enterprise service and ATL data sets. Thereby, we took our PES with the generalized assignment and a partition size of 100[13]. The results for the ATL data set demonstrated that intro-

[12]We restricted our data set to all samples with more than 200 elements. We also added examples with more than 800 elements of the complete ESR data with an F-Measure below 0.2 but above 0.1 to have an evaluation data set of 15 ESR mappings.

[13]We also tested 50, 150 and 200, where 100 was the most representative. However, the results and conclusion hold for the other sizes as well.



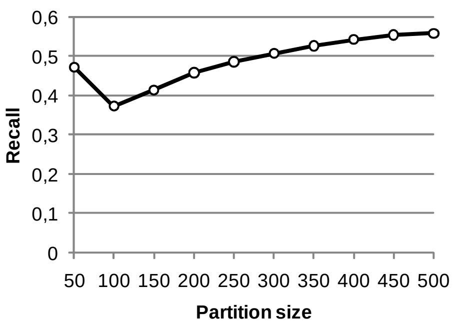


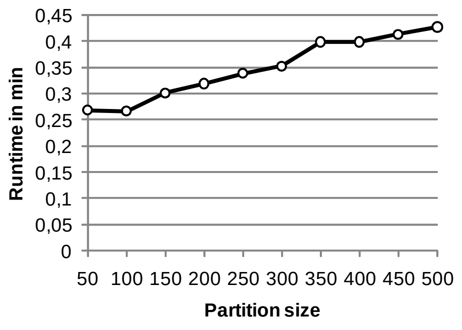
Figure 7.28: Comparison of precision and recall dependent on the partition size for PES

ducing partitioning creates an additional overhead, since the runtimes are 4.53 seconds with partitioning vs. 3.92 seconds without partitioning. It also shows a decrease in F-Measure by 4.49%. The decrease is reasoned by the removal of context information by partitioning a given input and matching those partitions independently.

**Minimal metamodel size** Our evaluation showed that partitioning should be applied for metamodels with more than 200 elements. Consequently, we also reduced the ESR data by removing examples where the participating metamodels have less than 200 elements, which leaves us with 15 mappings as gold-standards for evaluation.

**Result quality** Figure 7.28 depicts our measurements of result quality. Interestingly, the PES shows a precision of 0.27 and recall of 0.47 at a partition size of 50. This results from the structural arrangement of type information in the business schema metamodels. These types are isolated with their definition in partitions – by our PES and our proposed merging – and positively matched with each other. This information is lost when increasing the partition size because then more than the type information is contained in a single partition. The recall, as given in Fig. 7.28, shows also an increase beginning with a size of 50 up to 500, again due to the context information increase.

**Runtime and memory** We observe that the partition size influences the result quality. But it also affects memory and partitioning runtime as shown in Fig. 7.29. The PES shows a runtime of up to 0.43 minutes at worst. An increase in the partition size also leads to an increase in the runtime, because more re-partitioning and mergings have to be calculated. Regarding memory consumption, an increase in size does not affect the memory consumption



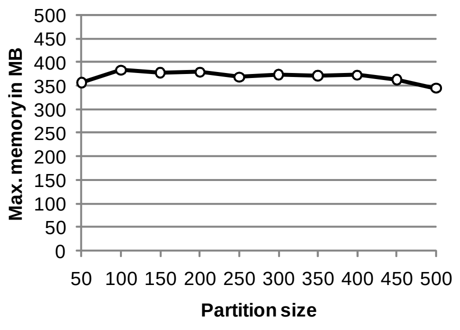


Figure 7.29: Comparison of memory and runtime dependent on the partition size for PES

much, because still the same number of elements and their partitions assignments have to be stored.

To summarize, the optimal trade-off in quality and scalability is given by a partition size of 50. The optimal values are precision 0.27, recall 0.47, runtime 0.27 min, and the memory consumption is 356 MB. Still, the quality and especially precision are lower than the values obtained by the original system, therefore we evaluated the assignment approaches in the following section.

### 7.6.2 Partition assignment

Based on our PES with a partition size of 50, we show precision, recall, F-measure, runtime, and memory consumption for all assignment approaches depending on their input parameter, e. g. a threshold for threshold-based assignment.

Thereby, we use the same configuration for the assigned partitions to be matched with the four baseline matchers: name, parent, children, and sibling. The threshold remains 0.3, with an average aggregation, but to reduce the number of wrong matches do not apply a delta selection for the single match tasks. The final output mappings are then combined using a delta of 0.04[14]. The similarity of partitions is calculated by 5 representatives. The representatives are chosen based on their degree, preferring the maximum.

#### 7.6.2.1 Threshold-based assignment

Investigating an increasing threshold from 0.1 up to 0.9 our first observation is no change for values between 0.1 and 0.5 (see Fig. 7.30 left). The reason for this observation is the partition similarity which exceeds these thresholds for partitions containing type information. At a threshold of 0.7 we can note

[14]Appendix B describes the aggregation and selection parameters in more detail.

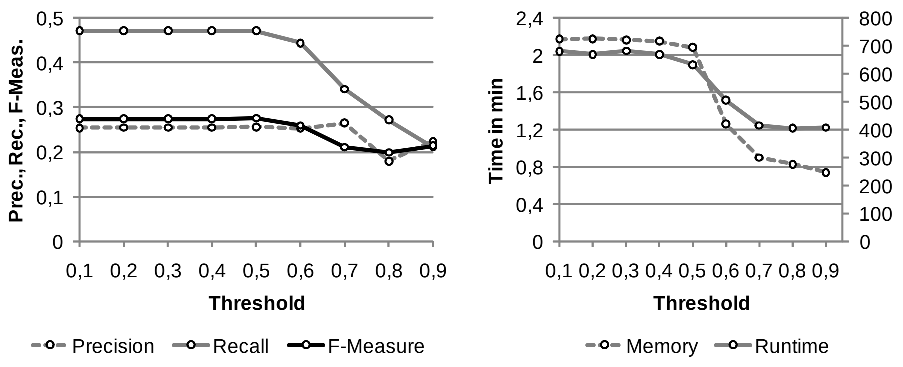


Figure 7.30: Quality, memory, and runtime results for different thresholds using threshold-based assignment

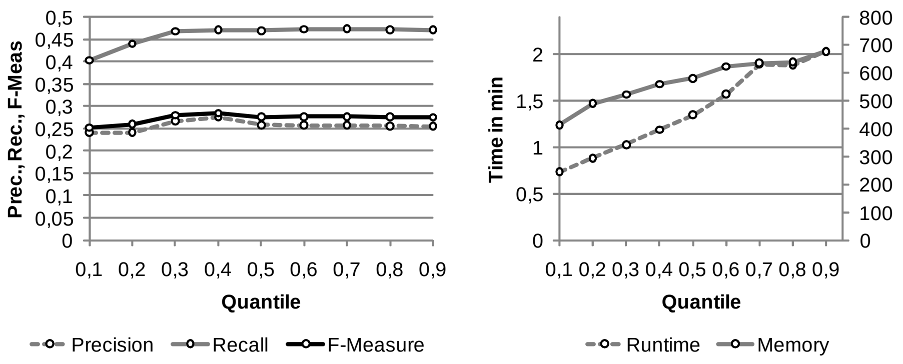


Figure 7.31: Quality, memory, and runtime results for different quantiles using quantile-based assignment

an increase in precision, followed by a decrease at a threshold of 0.8. This observation can be explained by the fact that for a high threshold a reduced number of partitions with low similarity are assigned to each other, which reduces the number of wrong matches leading to a high precision. But if the threshold is set too high correct matches are filtered out, decreasing the recall. The precision increase is accompanied by a decrease in recall, which is reasoned by the same argumentation, i. e. a reduced number of partitions matched also reduces the number of matches found.

The runtime and memory consumption are depicted in Fig. 7.30 on the right. The runtime and memory consumption decrease with an increase in the threshold, because fewer pairs are selected for matching and thus fewer resources are used. The best results in F-measure (0.259) for our data can be obtained by a threshold of $t = 0.5$. Considering the best quality the numbers for memory and runtime are 632 MB and 2.08 min.

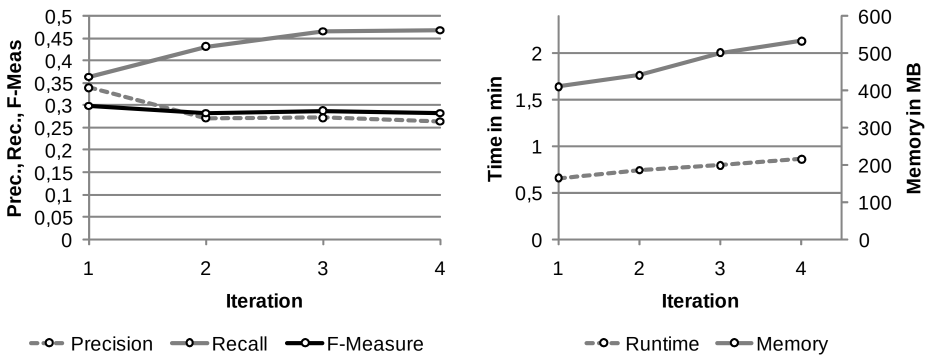


Figure 7.32: Quality, memory, and runtime results for different iterations using Hungarian assignment

#### 7.6.2.2 Quantile-based assignment

The quantile-based assignment is evaluated by increasing the quantile and thus the fraction of partitions to be matched. The results are shown in Fig. 7.31 with our quality measurements on the left. It can be seen that with an increasing number of partitions matched the recall increases while the precision slightly increases, at higher quantile values both remain unchanged. The explanation is similar to the one for threshold-based assignment, i. e. an increasing number of partitions matched means an increasing number of elements for matching. In contrast to threshold-based assignment there is no static threshold, but rather a dynamic threshold ensuring a minimum number of partitions to be matched. Accordingly, the memory as well as runtime increase for an increase in matched elements. The optimal F-Measure is given by a quantile of 0.4, i. e. 40% of all partitions are matched, leading to 559 MB in memory and 1.18 min in runtime.

In contrast to threshold-based assignment, the quantile approach leads to more stable results and is therefore the better choice for arbitrary matching tasks. In comparison to threshold-based assignment, quantile-based assignment shows a higher optimal precision and recall. The overall average F-measure for quantile-based assignment yields 0.284 compared to 0.259 for threshold.

#### 7.6.2.3 Hungarian

Aiming at optimal one-to-one partition assignments the Hungarian approach does not require defined parameters. However, the result of the algorithm may improve by the number of recalculations (iterations). The result for one up to four iterations is given in Fig. 7.32. The more iterations are executed the worse precision, runtime and memory consumption are.

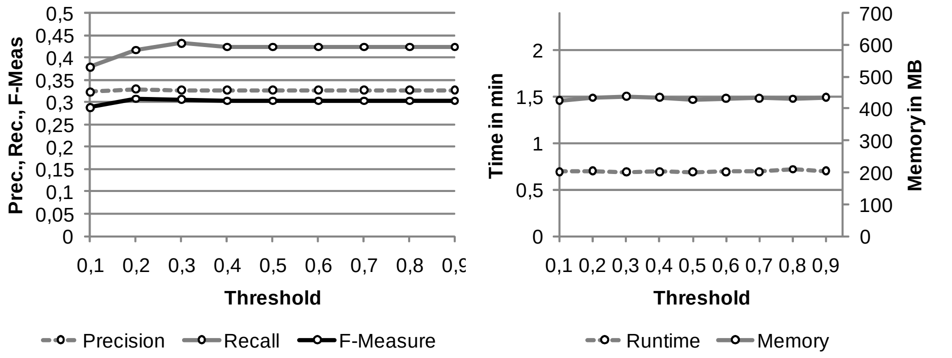


Figure 7.33: Quality, memory, and runtime results for different thresholds using generalized assignment

The maximal F-Measure of 0.298 can be obtained by executing one iteration. Thereby, the memory is 410 MB and the runtime is 0.65 min. The F-Measure is better than for threshold (0.259) and quantile (0.284). The average memory consumption of 410 MB is lower than for quantile (559 MB) and threshold (480 MB) because the Hungarian algorithm calculates optimal one-to-one assignments. Thereby, the memory used for the matching is only 181 MB, since the number of comparisons is reduced.

#### 7.6.2.4 Generalized

The generalized assignment can be investigated w.r.t. the internal threshold used for assignment of partitions. Thereby, we took values for a threshold between 0.1 and 0.9, in steps of 0.1. The results are depicted in Fig. 7.33 showing an indifferent behaviour. The memory consumption and runtime vary in a small interval around 430 MB and 0.7 min. The best F-Measure of 0.307 is given at a threshold of 0.2 with a memory of 438 MB and runtime of 0.69 min. Therefore, we chose this value for comparison with the other assignment approaches.

The result of our evaluation of the four assignment approaches is that threshold and quantile perform worse than Hungarian and generalized assignment which are similar. However, the generalized assignment has a higher recall of 0.42 compared to 0.36 of the Hungarian. In memory and runtime both assignment algorithms perform similar, thus based on our test data none of both can be favoured. However, the generalized assignment may perform better in case of many-to-many mappings in contrast to the Hungarian. Therefore, we chose the generalized assignment for a comparison to the baseline in the subsequent discussion.

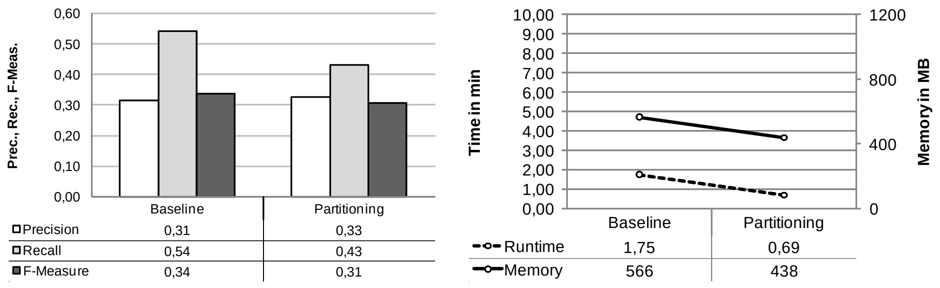


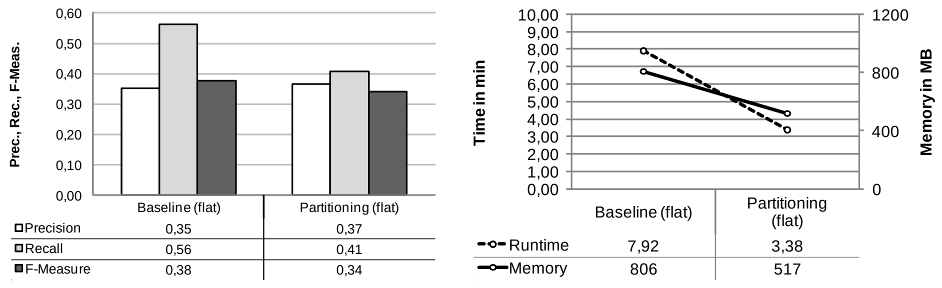
Figure 7.34: Quality, memory, and runtime results for partition based matching

### 7.6.3 Summary

We summarize our evaluation by comparing our baseline system MatchBox using the 15 large-scale metamodels with graph-based partitioning and generalized assignment for partition-based matching. Figure 7.34 depicts a comparison of the baseline to partition-based matching. We note that the overall runtime of a complete matching process can be reduced from 1.75 min to 0.69 min. The memory consumption can be decreased from 566 MB to 438 MB on average. These values were obtained for a local maching and no parallel matching. However, the concept provided by us also allows to match in parallel, even though we did not implement it. To summarize, on average 23% less memory is consumed and 60% less runtime is needed. Still, a small loss in quality (0.11 in recall) has to be accepted, which is partly compensated by a precision gain of 0.02.

Even though these numbers demonstrate the effectiveness we want to illustrate the scalability of our approach. Therefore, we changed our 15 large-scale examples and unfolded the graph by flattening it. That means we copied the content of type definitions to all referring classes, thus raising the size from about 1,400 elements to about 8,000 elements. The results for quality, memory, and runtime are depicted in Fig. 7.35. The quality shows the same behaviour, improving the precision by 0.02 where the recall is decreased by 0.15. However, the memory can be decreased from 806 MB to 517 MB on average and even more from 1,7 GB to 900 MB at maximum of the saving. The runtime is also improved from 7.92 min to 3.38 min. These numbers constitute savings of 36% in memory and 57% in runtime on average.

We effectively demonstrated that our planar graph-based partitioning approach reduces both runtime and memory consumption of a matching system. However, we also noted a limitation of our approach. It should not be applied to metamodels with fewer than 200 elements, because of the partitioning overhead.



| | Baseline (flat) | Partitioning (flat) |
|---|---|---|
| □Precision | 0,35 | 0,37 |
| □Recall | 0,56 | 0,41 |
| ■F-Measure | 0,38 | 0,34 |

| | Baseline (flat) | Partitioning (flat) |
|---|---|---|
| Runtime | 7,92 | 3,38 |
| Memory | 806 | 517 |

Figure 7.35: Quality, memory, and runtime results for partition based matching on flattened data

**Applicability** We can conclude that our approach is applicable to any metamodel with more than 200 elements.

## 7.7 Discussion of Results

Our evaluation investigated the improvements in quality by our GED matcher, by our mining matchers, as well as in scalability by our partitioning and partition assignment approaches. However, the observations made are related to the test data being used. As in all matching evaluations it is simply impossible to have a complete set of test data covering each case of data model possible. Still, our test data set of 51 examples with quite a diversity in structure and domain shows a range that has not been applied by other evaluations. Having discussed the base of our evaluation, we want to address in the following the applicability and limitations of our matching and partitioning approaches. Table 7.2 shows a summary of our evaluated concepts, their applicability, and their limitations.

### 7.7.1 Applicability

The first statement to be made is that we showed that our concepts are applicable in general. That means any metamodel can be processed by our algorithms, where planarity is enforced when not given. Thereby, for the real-world ESR at maximum 0.01 % of all edges are removed, which does not influence the result quality.

Next, our graph edit distance matcher can be applied to any metamodel to increase the result quality, especially in terms of precision. It has to be noted that the GED naturally performs the better the more the metamodels are structurally similar (source-target ratio). There are cases where the GED decreases the quality due to an input size mismatch and low token overlap, but they may be detected by a pre-processing of the input metamodels sizes

| Concept | Result quality | Applicability | Limitation |
|---|---|---|---|
| Tree-based matching | Baseline | Spanning tree of containment relations | Choice of pot. multiple trees |
| Planar graph edit distance | Increase | Metamodels | Size mismatch, token overlap needed, $k$-max degree only for ATL |
| Mining-based matching | Small increase | Non-identical metamodels | Small improvements (1%), Patterns need to exist |
| Planar partitioning | Decrease, Gain in memory and runtime | Metamodels of size $>$ 200 elements | Trade-off in recall and memory/runtime |

Table 7.2: Results and limitations for graph-based matching and partitioning

and token overlap. In case of a detected mismatch in size ratio and token overlap a fall back matcher can be applied.

The same applies to our graph mining matchers, which may be applied to any metamodel. However, to show an increase in result quality two requirements have to be fulfilled. First, the metamodels need to contain any patterns, and second the metamodels should not be identical. Again the second property can be easily checked by simple input metrics. The pattern existence can only be checked after the matcher has been applied, which may increase runtime. However, w.r.t. quality, if no pattern exists a fallback matcher may be used. Our observations show, that indeed the mining matchers do not increase the overall result quality. A possible reason is given by the pattern limitations by filtering, thus it may be worth to investigate how the search can be reduced before the mining, e. g. by light-weight matching techniques.

Regarding our mining matchers, it should be investigated how they can be applied in another way. For instance, by identifying patterns, mapping them, and reusing them in coverage approaches such as [132]. Another possible application is to apply pattern mining for metamodel decomposition and identifying mappable parts.

The partitioning we proposed is applicable for metamodels with more than 200 elements. The reason is that before reaching that size, partitioning is more expensive than matching and does not improve the results, thus partitioning should not be applied. Regarding large-scale metamodels we have shown that our partitioning can be applied reducing memory and runtime to more than a half, while improving precision with a small decrease in recall.

### 7.7.2 Limitations

One of the limitations of our structural approaches is the need for linguistic information. We observed, that relying only on structural information leads to a result quality decrease. However, combining linguistic and structural information leads to a result quality increase, especially in precision.

The next limitation is concerned with our $k$-max degree seed match approach for the GED. We observed that in case of the ATL-zoo result quality is increased using this approach. However, in case of the ESR data set it does not influence the quality at all. The reason is the behaviour of the GED which does not start at a given seed match, but rather reuses the seed similarity during computation. Beginning with the package element the $k$-max degree elements are either never reached during edit distance calculation or are already part of it. Therefore, to take advantage of them the GED would have to start at the seed match elements for the calculation.

In our implementation the GED does not start at a given seed match, because then it would have to start for multiple seeds at different points leading to multiple runs, which increases runtime considerably. Additionally, the separate results would have to be merged, a problem not investigated by us. Therefore, we see this as a point for further work.

Our graph-based partitioning leads to a loss in result quality, which shows the trade-off for a decomposition of a matching problem. One solution to this approach is an increase in the partition size, which comes along with an increase in memory consumption and runtime. Having knowledge of the mapping application and constraints one may choose a suitable size for a given matching problem.

## 7.8 Summary

In this chapter we presented our fourth contribution, a comprehensive evaluation of our proposed graph-based matching and partitioning approaches. We defined as our success criteria an increase of result quality and support for scalability. The approach we followed is thereby a comparison of a baseline system to the same system enhanced by our algorithms. Therefore, we first implemented a state-of-the-art matching system adapting tree-based schema matching techniques for graphs. Next, we described our test data sets from the MDA community and SAP. Based on these data sets we investigated the correctness and completeness of the results obtained using our approaches, i. e. precision and recall. For our graph-based matcher we observed a quality gain, while the graph-based mining only leads to minor improvements. Finally, we investigated the improvements in memory and runtime and the effect on quality for partitioning of large-scale metamodels demonstrating that we successfully meet the success criteria.

**Data sets** Our test data sets consist of 20 metamodels and mappings from the MDA community (ATL) and 31 SAP message mappings (ESR). Both data sets are made of heterogeneous data from different domains, such as purchase orders, UML version mappings, etc. We applied three state of the art matching systems on the data sets showing that indeed the mapping result quality shows an average F-Measure of 0.48. The F-Measures show that our 51 mappings are a hard matching task. It also shows the need for improvement in result quality.

**Planar graph edit distance** The planar graph edit distance matcher (GED) has been evaluated by us showing an improvement in result quality of up to 0.45 (45%) in precision. On average our GED improves precision by 7% resulting in an average increase in F-Measure by 4%. Thereby, it can be applied to any graph, but decreases in result quality with an increase in size mismatch between the two inputs. Furthermore, we have shown that the best results are achieved by separating reference- and inheritance-based matching.

**Graph mining** The graph mining matchers proposed by us were able to identify patterns in both data sets and derive corresponding mappings. However, they only increased the overall result quality by about 1%. The main reason for the low gain is the filtering of patterns to achieve a reasonable runtime. In addition, only some metamodels contain design patterns or redundant information that can be used by our mining.

**Planar graph-based partitioning** We have shown, that our planar partition-based matching (PES) reduces memory by about 23% and runtime by about 57%, while providing a gain in precision of 2% and loss in recall of 11%. Thereby, the memory consumption is reduced by the PES, while runtime improvements are achieved by selecting partitions to be matched based on the general assignment. The minimal size for a metamodel to be partitioned is 200 elements, while the optimal size of a single partition using our PES is 50.

# Chapter 8

# Conclusion

In this chapter we complete this thesis by presenting a summary and conclusions of our work. We revisit our contributions and derive recommendations for data model development from our results. Finally, we discuss and point out further research directions for graph-based matching and applications of planarity.

## 8.1 Summary

In the beginning of our thesis we established the basis for our work by introducing the fundamental concepts of metamodel matching and graph theory. In metamodel matching we defined a metamodel to be a modelling paradigm based on object-oriented concepts to describe a domain of interest. As a means of integrating heterogeneous metamodels we described the foundations of metamodel matching. We gave an overview of established element-level and structure-level matching techniques. Subsequently, we defined graphs and the related concepts of graph isomorphism calculation. To circumvent the complexity problem of graph isomorphism calculation, we took advantage of the special graph property of planarity. Planarity allows for reduced complexity of the isomorphism algorithms. We also introduced basic concepts and the state of the art in graph mining and graph partitioning.

Next, having defined the concepts used, we carried out the problem analysis of our thesis (Chap. 3). Applying the ZOPP methodology we defined our two cause problems of (1) insufficient correctness and completeness of matching results as well as (2) insufficient support for large-scale metamodel matching. We then performed a root-cause analysis for these two problems, identified the scope and derived the following objectives: exploit structural information for matching, exploit redundant information for matching, and support matching of metamodels of arbitrary size. These three objectives impose requirements which lead to our research question of how to improve metamodel matching by graph-based algorithms. We also presented our research approach which is to analyse and adopt existing graph theory algorithms, rather than to develop new algorithms from scratch.

In Chap. 4 we investigated related work in the area of large-scale metamodel matching. Thereby, we studied the three areas of metamodel, ontology, and schema matching. We introduced related matching systems and algorithms in each of those areas especially pointing out approaches which tackle matching quality and scalability. We compared the related matching systems and matching techniques, and concluded that most of the approaches use tree-based techniques or similarity flooding and none of them apply global graph-based matching with planarity. The scalability problem is tackled only by a few approaches in schema and ontology matching but none of them applies structure preserving partitioning and also none of them (explicitly) addresses different solutions for the partition assignment problem.

We presented our first two contributions of structural graph-based matchers in Chap. 5. We proposed the adaptation of a planar Graph Edit Distance algorithm (GED). The GED calculates similarities between two metamodel graphs in quadratic runtime complexity. Thereby, the GED has been extended by us to take linguistic and structural information into account. To improve the GED result quality we proposed a seed mapping approach named $k$-max degree, which makes use of $k$ user input mappings ranked according to their number of neighbours (degree).

We further presented our graph mining matchers based on the idea of discovering reoccurring patterns in a metamodel for matching. We proposed two matchers, the design pattern matcher and the redundancy matcher. The design pattern matcher mines for patterns based on an incremental extension of a pattern. The design patterns are found by incrementally checking two metamodels in parallel for valid patterns. The redundancy matcher we proposed mines two metamodels independently for patterns. The mining is based on the principle of reducibility, reducing the most frequent edge type until no further reduction is possible. The resulting patterns for both metamodels are then mapped using our proposed planar GED.

Next, we presented our third contribution, a graph-based partitioning algorithm for large-scale matching and a comparison of partition assignment algorithms (Chap. 6). We proposed to adapt an existing planar partitioning algorithm to solve the memory problems in the context of large-scale metamodel matching. The algorithm ensures partitions of similar size by an iterative splitting of an input metamodel. We extended the algorithm by proposing a merging phase based on coupling and cohesion, thus taking structural information into account. We also presented and discussed the threshold, quantile, Hungarian, and Generalized assignment algorithm with the goal of runtime reduction. In our evaluation we studied the behaviour of those assignments and made observations when to apply which one.

With our comprehensive evaluation (Chap. 7) of the graph-based metamodel matching and partitioning we provided the fourth contribution. We presented two data sets used for a comparison of our baseline system MatchBox to our algorithms. The test data consists of 51 gold-standard mappings and originates from the ATL-zoo and the Enterprise Service Repository. We investigated different metamodel graph representations with our test data and concluded that a graph based on containment relations performs best in quality for tree-based matching. Thereby, quality refers to the correctness (precision) and completeness (recall) of the mappings calculated. We observed a successful quality gain for the GED, while the graph-based mining only yielded minor improvements. Finally, we studied the memory and runtime of our baseline system MatchBox with and without our planar partitioning algorithm. The results with partitioning were a substantial reduction in memory consumption and runtime with a small trade-off in quality.

In the following section we will revisit our contributions, provide the numbers of our evaluation, and discuss the results obtained in more detail.

## 8.2 Conclusion and Contributions

In the introduction of our thesis we claimed to provide four contributions. In our problem analysis we derived two main objectives as well as our main research question. In the following we want show how they fulfil the objectives defined. This leads to an answer of our research question by interpreting our evaluation insights. In this thesis the four contributions presented were:

C1 The planar graph edit distance matcher for improvements in correctness and completeness by metamodel graph similarity calculation,

C2 The design pattern matcher and the redundancy matcher, utilizing redundant information and design patterns for improvements in correctness and completeness,

C3 The planar graph-based partitioning for metamodel matching and a study of assignment algorithms for reduction in runtime and memory consumption, and

C4 A comprehensive real-world evaluation of our approaches based on 51 industrial large-scale gold-standard mappings.

In the following two paragraphs, we will compare objectives 2 and 3 resulting from the main objective 1 *an increase in matching quality and support for scalability* (Sect. 3.2.2) to our contributions.

**Increase correctness and completeness of matching results** One main objective was to increase completeness of matching results. The subobjective which contributed largely to the main objective was to *2.2 exploit structural information for matching*. The structural information was exploited by our planar GED combining linguistic information and global graph-based structure. The resulting improvements were up to 45% in correctness (precision) and up to 27% in completeness (recall). On average the precision improved by 7%, the recall remained almost unchanged (-0.9%), and the F-Measure improved by 4%, which shows that the objective has been fulfilled.

The approach of subobjective 2.4 to *exploit redundant information for matching* provided a smaller contribution. The subobjective was achieved by our design pattern matcher and our redundancy matcher. Both matchers show small improvements on average with 2.2% in precision, 1.2% in recall, and 1.5% in F-Measure and are limited to examples which show patterns. Therefore, the two mining approaches contributed considerably less to the objective 2.

**Support matching metamodels of large-scale size** This objective addresses the scalability support for matching. The objective has been split into three subobjectives, one targeting the memory consumption, one the runtime, and the last the quality trade-off.

Subobjective *3.1 a concept for managed memory consumption* requires us to reduce the memory consumption of the overall matching process. This is tackled by our planar partitioning. The partitioning produces similar-sized partitions under the restriction of an upper bound on the partition size. An independent matching of these partitions ensures a managed memory consumption. Our evaluation showed that with a partition size of 50 elements on average 23% of memory is saved. We also presented a concept which allows for structure-preserving distributed matching of metamodels of arbitrary size; hence we consider the objective as fulfilled.

The second subobjective was *3.2 a concept for reducing matching runtime* which we pursued by a study of our assignment algorithms for partitions. We concluded with the generalized assignment algorithm as an optimal choice with an average precision gain of 2%. By assigning partitions to be matched with each other we reduced the number of comparisons which has been confirmed by our evaluation, demonstrating a runtime reduction of 57% on average, compared to traditional approaches.

However, especially the memory reduction does also reduce the quality, because of the reduced matching context. The third subobjective thus required to *3.3 optimize the trade-off between scalability and matching result quality*. In our evaluation we evaluated best parameters for partition size, assignment algorithms, etc. to conclude that by a size of 50, i. e. an average type size of our test data, and the generalized assignment the best trade-off is given. Consequently, we also fulfilled this objective.

With the fulfilment of our two main objectives we reached our overall goal of increasing matching quality and supporting scalability. Moreover, we also enabled distributed matching with our planar partitioning, thus removing any memory boundaries. This allows to match large-scale metamodels in a cloud-based environment and thus in a distributed manner, e. g. as also under investigation by [50, 125]. In addition, we also demonstrated the feasibility of planarity and thus answered our three hypotheses, which we will now discuss in more detail.

**Research question and hypotheses** The first hypothesis *H1* stated that planar subgraph isomorphism calculation improves result quality. This hypothesis has been confirmed by our planar GED (C1) and our evaluation (C4). In almost all cases, the GED utilized global structural and linguistic information to improve the matching result quality confirming H1.

The hypothesis *H2* stated that the mining on metamodel graphs improves result quality. This hypothesis has been confirmed for some cases by our evaluation (C4) and our two mining matchers (C2). In case of existing patterns the mining matcher can improve the result quality. However, in most cases no patterns were found and thus the result quality remained unchanged. Therefore, we can only partly confirm our hypothesis and see room for further work in mining-based matching.

The third and last hypothesis *H3* stated that planar partitioning can provide support for large-scale metamodel matching. Our evaluation (C4) and the planar partitioning with assignment (C3) demonstrated that indeed the hypothesis is correct.

Overall, two of our hypotheses have been fully confirmed and one partly. We conclude that we reached our goal and improved the matching result quality and also provided support for large-scale metamodel matching. Therefore, our thesis answered our research question: ”How can structural graph-based algorithms improve the correctness, completeness, runtime, and memory consumption of metamodel matching?”.

Still, not all of our hypotheses could be confirmed and some limitations exist. In the following section we will make recommendations for data model development that would allow exploiting information contained in a metamodel for improved matching.

## 8.3 Recommendations for Data Model Development

Our evaluation pointed out that even when applying graph-based techniques the resulting quality may be insufficient. In addition, we presented a concept for scalability support but with a trade-off in quality. Therefore, we want to present recommendations for data model design such that a resulting data model provides more information for improved matching.

**Planar modelling** To maximize the performance of the matching algorithms a metamodel should fulfil our basic assumptions of planarity. Although data model designers tend to create planar models they do not ensure it, because in some cases complex non-planar relations are needed. We recommend to create planar models to allow taking full advantage of the structural information of a metamodel.

**Precise modelling** Precise modelling refers to recommendations to prevent modelling of untyped elements and apply domain-specific modelling. The problem of untyped elements is present when element type semantics are modelled using string attributes, e. g. for complex types or enumerations. These implicit semantics are hard to match and hence should be avoided by modelling types as explicit classes and enumerations as such, thus being precise.

Imprecise modelling also concerns the modelling of redundant information rather than modelling types. This information may be used by our redundancy matcher, but we recommend avoiding such information. Instead, modelling of types and design patterns should be applied, because such information is easy to match and allows for reuse.

The modelling of types introduces another problem that is a mixed modelling of domain-specific and unspecific parts which are hard to match. We recommend being as domain-specific (precise) as possible. The more specific the information is modelled, the better a matcher can distinguish between elements and thus find similarities. This is supported by a separation of domain-specific and unspecific information via modules, because they allow for an explicit matching context.

**Modularization** A problem encountered are huge metamodels without any separation into modules. Their size leads to an immense search space for matching techniques as well as a splitting of connected big types in case of partitioning, because connected types are elements forming a composite element by containment relations. The partitioning of such types leads to worse results, because only parts are matched. Our recommendation is to separate a metamodel using hierarchical modules (packages). These modules ideally contain 50 elements, complying with our partition size. The module hierarchy supports our partitioning due to the level selection approach which aims at hierarchical models.

When assigning the partitions, the representatives we select for the partition similarity potentially provide insufficient information, and in this case the quality of the matching is reduced due to missing or incorrect assignments. Consequently, we recommend to model module representatives with a precise labelling, i. e. a module root element or elements which act as dedicated representatives and may lead to more correct assignments.

**Reuse** We also advocate reuse of information, because computational runtime is needed if multiple reoccurring information has to be identified, e. g. by our redundancy matcher, and this information may be incorrect. It would be more efficient to put these shared elements into modules and reuse these modules across metamodels. These modules would allow for improved matching results, e. g. by reuse-based approaches such as [132].

The recommendations given are design guidelines for a metamodel designer in order to support matching. However, even if a perfect metamodel is given, due to the insufficient expressiveness of metamodels there is still further work to be done in matching. This point among others is outlined in the subsequent section on further work.

## 8.4 Further Work

We present directions for future research on metamodel matching in general, our matching and partitioning techniques, and algorithms making use of planarity.

**Metamodel matching** The quality problem in metamodel matching still exists, as it is the case in the areas of schema and ontology matching. Therefore, it is still necessary to either change the matching process or research on matching techniques for improved result quality w.r.t. a certain domain or problem.

Nowadays, matching assumes a complete matching calculation and presentation to the user. An alternative matching approach could consider the low result quality and instead present top-n matches to the user. In the area of schema matching one approach on top-n matches has been proposed [42]. However, research on a guided process, possible incremental learning, and ways of visualizing is still needed.

Interestingly, our investigation of the state of the art also revealed that reuse-based and statistical matching have not been researched intensively. Consequently, they constitute one direction for further research, for instance techniques from model matching [81, 82] may be adopted for that purpose.

Another open issue is the upper bound for the average F-Measure. A formal analysis of the matching problem and techniques may show or prove the upper bound for matching, for instance w.r.t. certain data set characteristics. If an upper bound can be formally determined it is possible to provide users of matching tools statements on the quality of the mappings automatically calculated.

This goes along with the next problem of a missing metamodel matching benchmark, which would allow for a tool and technique comparison. A benchmark could also provide hints for promising techniques for adoption and insights into an average matching quality. A possible benchmark could benefit from the ATL-zoo data as proposed by us in [151] and App. A.1, because it is publicly available.

**Planar GED** Our planar Graph Edit Distance matcher (GED) provides gains in quality but also shows drawbacks for size mismatches and low token overlap of source and target metamodels. This problem may be tackled by a heuristic approach which applies a light-weight matching technique and starting at the most similar element. In addition, $k$-max degree seeds as starting point for the edit distance calculation are also promising. This procedure may be repeated for multiple $k$-max seeds or light-weight matches. This opens the question for a merging of the results of multiple GED runs which has to be answered by further research.

We also observed the precision improvements of our GED. Since this indicates a filter behaviour of our matcher it may be promising to integrate it into a two-staged process. The first stage calculates mappings based on local techniques, e. g. a name and parent matcher, where the second stage refines the results obtained. To achieve this it has to be researched how to combine our GED with a matching process-based system, e. g. as in [123].

The GED proposed by us is currently limited to metamodel matching, because it uses metamodel specific type information for two separate runs (reference and inheritance). However, it is interesting to see how the GED behaves when applied to arbitrary models. Such work could yield promising results as outlined by us in [148], because traditional model matching techniques rely on labels and statistics less on structure [81].

**Mining-based matching** The mining-based matching has shown a less favourable matching quality in the average case. The cause problem is the exploding search space and thus runtime and memory consumption. Therefore, more research is needed on how to prune and decrease the initial search space. One possible approach to be taken is to restrict the search space to pre-defined patterns [130], but this restriction induces quality problems due to domain-specific patterns. Another solution to the search space problem can be given by a pre-processing phase based on a preselection of relevant regions via partitioning or light-weight matchers. This allows for an increased pattern size due to search space reduction, while still not limiting the space to pre-defined patterns.

It is further interesting to investigate how pattern mining can be applied for metamodel decomposition and mapping reuse, because the problem of mapping reuse approaches, e. g. as in [132], is the initial pattern repository. Using our mining, frequent patterns can be extracted and stored in a repository, and then a mapping reuse approach can be applied to derive new mappings.

**Partition-based matching** The most obvious missing work to be done for our partitioning is an implementation of distributed matching using our partitioning. To this point we only provided the concept for distributed and parallel matching but did not implement it. If implemented, it is especially interesting to observe specific match configurations for certain partition pairings.

So far we apply the same matchers and configuration for each partition pair. However, particular partition pairs may require particular matchers and configurations. Therefore, it should be studied how to derive the matchers and configurations for partition pairs. When the configurations are available it may be interesting to research matching task distribution on multiple machines w.r.t. matcher configurations.

Another open point is the partition similarity calculation. The calculation time and assignment quality may be increased by applying an incremental approach. Instead of comparing each pair of representative elements the process could change to perform assignments during the similarity calculation. For instance, partitions are assigned based on the maximal similarity. However, it needs to be researched how to process the partition pairs exactly. Also the worst case behaviour of identical partition similarity needs to be studied.

**Planarity** With our thesis we successfully applied graph algorithms based on planarity. We observed that the graph theory of the last century provides promising algorithms for a multitude of problems.

Planarity as a special graph property turned out to be useful for metamodel matching. However, it may also be used for other algorithms and applications, when it allows for more information available for calculation, e. g. for software diagram layouting [71] or any problem connected to shortest path calculations under special assumptions as in [80].

**The end** With our thesis we provide contributions to the field of metamodel matching in terms of quality and scalability. We encouraged the use of the graph structure and of the rather unused graph property planarity. Although our partitioning solves the scalability problem via distributed matching, structural matching still does not solve the quality issues completely, thus the never-ending journey on matching quality continues.

# Appendix A

# Evaluation Data Import

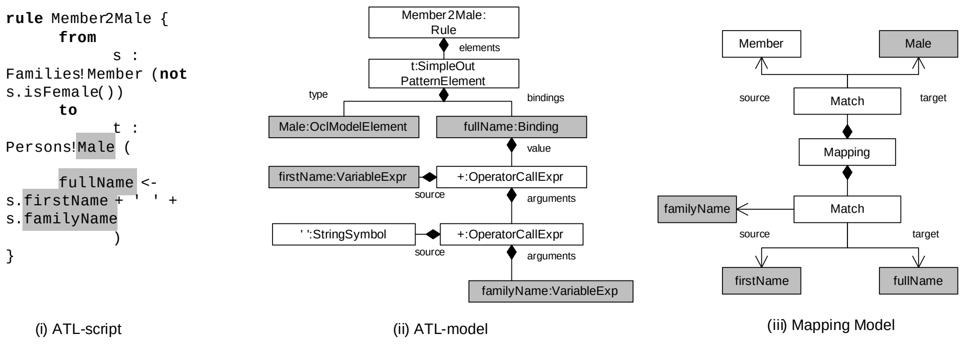
## A.1 ATL-zoo Data Import

We propose to use existing model transformations as gold standards for matching evaluation. Since an evaluation by model transformations requires a considerable amount of test data, one has to choose a model transformation providing such data. Once the data is available, the transformation has to be imported into a mapping serving as a base for comparison.

The idea is to use a common mapping metamodel (format) to compare the result mapping of a matching system and the gold-standard. For the gold-standard mappings we used the Atlas Transformation Language (ATL) [70], which is a model transformation language providing the base for a considerable range of 103 transformations in the ATL-zoo[1]. Our approach is based on the matching component MatchBox, a matcher combination system as proposed in [151].

Figure A.1 depicts an example of an ATL-script import into the ATL-model and finally into the mapping metamodel of MatchBox. The exam-

[1]`http://www.eclipse.org/atl/zoo`



```
rule Member2Male {
      from
            s :
Families!Member (not
s.isFemale())
      to
            t :
Persons!Male (

      fullName <-
s.firstName + ' ' +
s.familyName
            )
}
```

Figure A.1: Example of an import from ATL

ple presents an ATL-transformation between two different metamodels describing a family. The metamodels can be described as follows. The first metamodel defines a family as *Member*(s) differentiating on the gender by a boolean attribute. In contrast, the second metamodel separates the concepts by the explicit classes *Male* and *Female*. The names of persons are also modelled differently, the first uses a *firstName* and *familyName*; the second uses the *fullName*.

Figure A.1 (i) depicts the ATL-script transformation between both metamodels. The parts highlighted in grey show corresponding elements. The `from` and `to` parts define the (declarative mapping) between both classes *Member* and *Male*. The part defining the name mapping is the (imperative) mapping in the so-called binding. Figure A.1 (ii) shows the corresponding ATL-model and (iii) the representation in our internal mapping metamodel.

In the following sections we will describe the transformations between the ATL and MatchBox mapping model representations.

### A.1.1 Import of metamodels

As discussed before, the imported metamodels' elements are necessary for a creation of a mapping model. Therefore, the metamodel importer is responsible for traversing the metamodel elements and publishing them in the internal mapping metamodel. In our implementation we applied a graph-based model as utilized by MatchBox. The internal model is a subset of EMOF and therefore represents a typed graph that supports inheritance and associations as first-class concepts.

### A.1.2 Import of ATL-transformations

The import of an ATL-script is twofold. First, we transform the ATL-script into an ATL-model. Second, we apply a model transformation for creating a corresponding mapping model. The actual import is a mapping between the ATL metamodel as in Fig. A.2 and our mapping metamodel as in Fig. A.3.

A model transformation (module) in ATL is a set of transformation rules. These transformation rules are the declarative part of ATL. ATL also offers an imperative part, i. e. a so-called binding of a transformation rule and a helper expression. Consequently, the following parts are considered:

- A *Declarative mapping* consists of an in- and out pattern defining corresponding classes.
- An *Imperative mapping* consists of bindings and helpers specifying expressions on corresponding attributes.

The internal mapping metamodel of MatchBox is simpler, because it is not executable. A mapping consists of a set of matches and each match

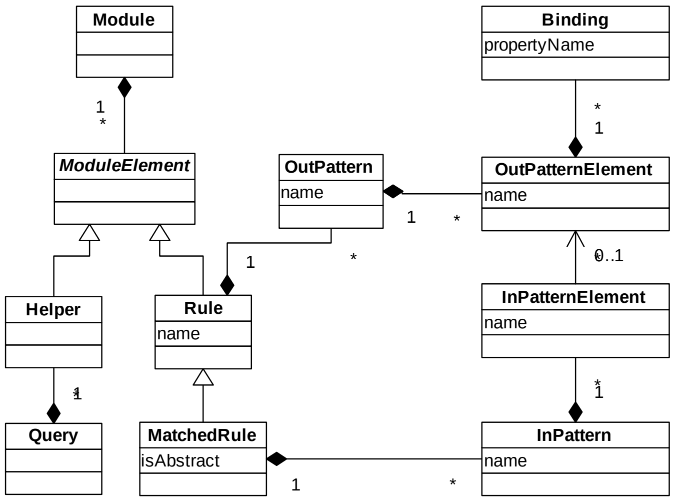


Figure A.2: Excerpt of the ATL metamodel

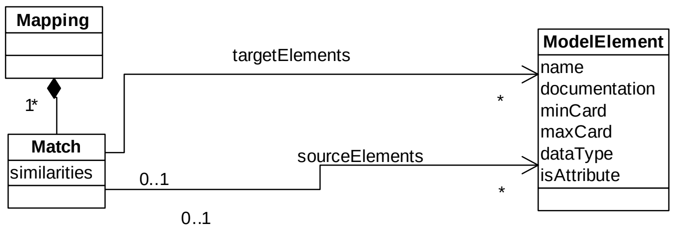


Figure A.3: Excerpt of the MatchBox model

defines a correspondence between a set of source and target element, as given in Fig. A.3.

The mapping between both models is shown in Table 1, there we describe an ATL concept and the corresponding MatchBox mapping model concept providing a description for further details or restrictions. Subsequently, we give a description of the mapping grouped into the declarative and imperative mapping.

**Declarative mapping** An ATL-transformation is the starting point of our import. This transformation contains any number of rules that constitutes the transformation. Each rule consists of an in- and an out-pattern. The in-pattern denotes the source element and the out-pattern the target element(s). Our mapping metamodel consists of a mapping, followed by a set of matches, each relating source elements and target elements. Consequently, a mapping is created for the transformation, whereas each transformation rules' components lead to a match. Algorithm A.1 shows the implementation of the mapping mentioned before in pseudo code.

**Algorithm A.1** ATL-Rule Import

```
Require: $M_{atl}, M_{map} \in Model, m \in Match$
 1: resolve all alias and helper definitions
 2: for all $r$ in $M_{atl}.rules$ do
 3:     for all $out$ in $r.outs$ do
 4:         $m \leftarrow$ create $m$
 5:         $m.source \leftarrow$ get $S(r.in.type)$
 6:         $m.target \leftarrow$ get $S(r.out.type)$
 7:         importBinding($r.binding$)
 8:         $M_{mapping} \leftarrow M_{mapping} \cup m$
 9:     end for
10: end for
```

$M_{atl}, M_{map}$ are the input ATL-model and the mapping model and $m$ is the output match returned by the rule import. Since ATL supports the definition of so-called helpers, they have to be resolved as well. This is done by using a helper's signature, i. e. the return type and the input parameters. This constitutes a mapping from the input to the output. Afterwards all rules are traversed by creating a match which gets as source and target the corresponding in and out elements. Besides these explicit elements, a rule consists of a binding, which specifies constraints over the in- and out-patterns. For instance, a *Binding* specifies a mapping between attributes of the in- and out-patterns. Therefore, each binding has to be evaluated too.

Table A.1: Mapping from ATL to the MatchBox mapping metamodel

| ATL Element | MatchBox Element | Description |
|---|---|---|
| Module | Mapping | – |
| Module.elements | Mapping.matches | – |
| Rule | Match(es) | – |
| Rule.outPattern.elements | match.target | Each combination of in- and out-pattern is mapped onto match |
| Rule.inPattern.elements | match.source | Each combination of in- and out-pattern is mapped onto match |
| Binding.property | match.target | A binding's property is mapped onto a target of a match |
| Binding.value | match.source/target | The value is mapped depending on its type as specified below |
| Nav.OrAttr.CallExp | match.source | Resolving the expression a match source is determined, whereas the binding's property is set as a target of the match |
| VariableExp | match.target | The resolved type of the VariableExp is mapped onto the match target using the in pattern as source element(s) |
| OperatorCallExp | match.target | Each operand is mapped onto a new match using the in pattern sources and mapping according to the operand's type |
| SequenceExp | match.source | Each element of the sequence is mapped as defined before using it as a source for a match, whereas the bindings property is set as a target of the match |

**Algorithm A.2** Binding Import (importB)

```
Require: $b \in Binding, m \in Match$
 1: $v \leftarrow b.value$
 2: if $v$ isOfType NavigationOrAttributeCallExp then
 3:     $m \leftarrow$ importB $(v.source, v)$ {resolve type definitions}
 4: else if $v$ isOfType VariableExp then
 5:     $m \leftarrow$ resolve variable type of v.referredVariable and get corresponding
        model element
 6: else if $v$ isOfType OperatorCallExp then
 7:     $m \leftarrow$ importB $v.value$ {resolve parameters and types}
 8: else if $v$ isOfType SequenceExp then
 9:     for all $e$ in $v.value.elements$ do
10:         if $e$ isOfType NavigationOrAttributeCallExp then
11:             $m \leftarrow$ importB $e$
12:         end if
13:     end for
14: else
15:     {do nothing}
16: end if
```

**Imperative mapping** Our *Binding* evaluation is depicted in Algorithm A.2. A binding has a so-called value, which is an *OclExpression*. This OclExpression can have multiple types. However, we consider a limited range that covers the most common ones. A binding's value could be either a *(1) NavigationOrAttributeCallExp*, for instance `rule.name` or a *(2) VariableExp*, e. g. `A`. A binding is also allowed to be an *(3) OperatorCallExp*, e. g. `not` or a *(4) SequenceExp*, e. g. `a + b`. A binding also refers to a specific property which has to be resolved according to the context provided by the out-pattern. In the following we will describe each of these cases.

The *(1) NavigationOrAttributeCallExp* (line 2 to 3) denotes the '.' operator, thus referring to a specific attribute which has to be resolved in order to constitute a mapping. Once the type of this attribute is available, it is mapped on the target element determined by the binding property and a corresponding match is created. Referring to our example in Fig. A.1 the `s.firstName` denotes a *NavigationOrAttributeCallExp*, whereas the `firstName` is a *VariableExp*.

A *(2) VariableExp* (line 4 to 5) is imported by resolving the type of the variable this expression refers to. The match is created from the type for the previous target. In our example in Fig. A.1 the `firstName` and `familyName` are of type *VariableExp*.

Importing an *(3) OperatorCallExp* (line 6 to 8) leads to a handling of the concatenation operation, all other operations are ignored. Thereby, all operands are resolved regarding their type and for each operand a match is created. This match target is the one from the beginning.

Figure A.1 also depicts an example for a *(4) SequenceExp* which is the + in `s.firstName + ' ' + s.familyName`. This results in an evaluation of both participating expressions of the names.

We implemented the import recursively, because these expressions can be nested, e. g. the import of a *VariableExp* can trigger the import of another *VariableExp*.

## A.2 ESR Data Import

Below we provide the EMFText grammar used for the parser generation. This parser imports the ESR data into an internal representation, which again is transformed into the internal mapping metamodel of MatchBox.

```
SYNTAXDEF es
FOR <http://emftext.org/enterpriseServices>
    <EnterpriseServiceMapping.genmodel>

START Mapping
OPTIONS {
    reloadGeneratorModel = "true";
    generateCodeFromGeneratorModel ="true";
    usePredefinedTokens = "true";
    memoize ="true";
}

TOKENS {
    DEFINE NAMEPATH $ ('/'  ('0'..'9'|'a'..'z'|'A'..'Z'|'_'|'/'|
        '@' | '[' | ']' |'.' | ':' | '-')*) $;
    DEFINE EXPRESSION $ (('0'..'9'|'a'..'z'|'A'..'Z'|'_' | '\'' | '@' |':')
        ('0'..'9'|'a'..'z'|'A'..'Z'|'_'|'/'| '@' |'\'' | '.' | ':' |
        '-' | '=' | '{' | '}' | '?' | '@' )*) $;
    DEFINE OPERATOR $ (('a'..'z' | '0'..'9'| 'A' ..'Z')*  '(')$;
}

RULES {
    Mapping ::= matches*;
    Matches ::= target "=" source;
    ConstExpression ::= ("getSendSys()" | "sender()" |
        "const(value=" (value[EXPRESSION])? ")" );
    ExpressionList ::=  expression ("," tail)? ;
    AttributeList ::=  ("," |"%WHITESPACE")? attribute[EXPRESSION]
        ("," tail)? ;
    SourceSchemaElement ::= (namepath[NAMEPATH] | "null") ;
    Target ::= namepath[NAMEPATH];


    ConcatExpression ::= "concat(" elementList? ",
        delimeter=" (delimiter[EXPRESSION])? ")";
    ConcatLinesExpression ::= "concatLines(" elements ("," options)? ")";
TrimExpression ::= "trim(" source ")";
SubstringExpression ::= "substring(" source ("," options)? ")";
SplitTextExpression ::="splitText(" elements ("," options)? ")";
ToUppercaseExpression ::="toUpperCase(" source ("," options )? ")";

TransformDateExpression ::= "TransformDate(" source "," options ")";
CurrentDateExpression::= "currentDate(" options ")";

CreateIfExpression ::="createIf(" condition ")";

FixedValueExpression ::= "FixValues(" source ("," table)? ")";
CopyPerValueExpression ::= "CopyPerValue(" elements ("," options)? ")";
SplitByValueExpression ::= "SplitByValue(" source ( "," options)? ")";
ReplaceValueExpression ::= "replaceValue(" source ( "," options)? ")";
CopyValueExpression ::= "CopyValue(" source ("," options)? ")";

SumExpression ::= "sum(" source ("," options)? ")";
CountByTemplateExpression ::= "countByTemplate("elementList
    ( "," options )? ")";
CountExpression ::= "count(" source ( "," options)? ")";
CounterExpression ::= "counter(" options ")";
GetPredecessorExpression ::= "GetPredecessor(" elements
    ("," options)? ")";

RemoveContextExpression ::= "removeContexts(" source ")";
CollapseContext ::= "collapseContexts(" source ")";

ExistsExpression ::= "exists(" source ")";
AndExpression ::= "and(" source1 "," source2 ")";
OrExpression ::= "or(" source1 "," source2 ")";
NotExpression ::= "not("  source ")";
IfExpression ::= "iF(" elements ( "," options )? ")";
IfWithoutElseExpression ::= "ifWithoutElse(" elements ( "," options )?
    ")";
StringEqualsExpression ::= "stringEquals(" leftSide "," rightSide ")";
EqualsAExpression ::= "equalsA(" leftSide "," rightSide ")";
CheckAllTrueExpression ::= "checkAllTrue(" elements ( "," options )? ")";
CheckIfEmptyExpression ::= "checkIfEmpty(" source ( "," options )? ")";
CheckTrueExpression ::= "CheckTrue(" source ("," options )? ")";
CheckForTrueExpression ::= "checkForTrue(" elements ("," options)? ")";
ConvertToStringExpression ::= "convertToString(" (source ",")?
    options  ")";
GenerateItemIDExpression ::= "generateItemId(" source ("," options)?
    ")";
MapWithDefaultExpression ::= "mapWithDefault(" source ("," options)?
    ")";
ValueMapExpression ::= "valuemap(" source ("," options) ? ")";
UseOneAsManyExpression ::="useOneAsMany(" elements ("," options)? ")";
AddFormatExpression ::= "addFormat(" elements ("," options) ? ")";
FormatByExampleExpression ::= "formatByExample(" elements
    ("," options)? ")";
UnknownOperationExpression ::= operation[OPERATOR]  elements
    ( "," options )? ")";
```

# Appendix B

# MatchBox Architecture

## B.1 Architecture

Figure A.1 shows an overview of MatchBox's architecture. As described in Sec. 7.2 it uses an exchangeable matching core, which includes a set of matching components (matchers) operating on the information provided by an internal representation of metamodels. Additionally, it contains a combination component providing means for an aggregation and selection of results. The core, configuration and combination component are implemented by the AMC.

The matching core operates on the repository which contains all metamodels, provided models (instances) and mappings. The mapping importer implements an import from the ATL transformation language, the Ontology Alignment API (OAA) [34]), and the ESR mapping format to the mapping model of the AMC. Thereby, the internal data model is a typed graph called Genie. For a detailed description see the master thesis in [67].

The configuration component implements the configuration of the aggregation and selection strategies. The metamodel importer transforms metamodels into the internal data model. The data model of the repository imposed by the AMC matchers is introduced in the subsequent section. We have implemented the MatchBox prototype using Java and the Eclipse Modeling Framework (EMF) [109]. MatchBox's repository is consequently EMF-based; the core etc. is implemented in Java.

## B.2 Combination Methods

- **Aggregation** The aggregation reduces the similarity cube to a matrix, by aggregating all matcher's results matrices into one. This aggregation is defined by four strategies: *Max*, *Min*, *Average*, and *Weighted*. The *Max* strategy is an optimistic one, selecting the highest similarity value calculated by any matcher. Contrary, the *Min* strategy selects the

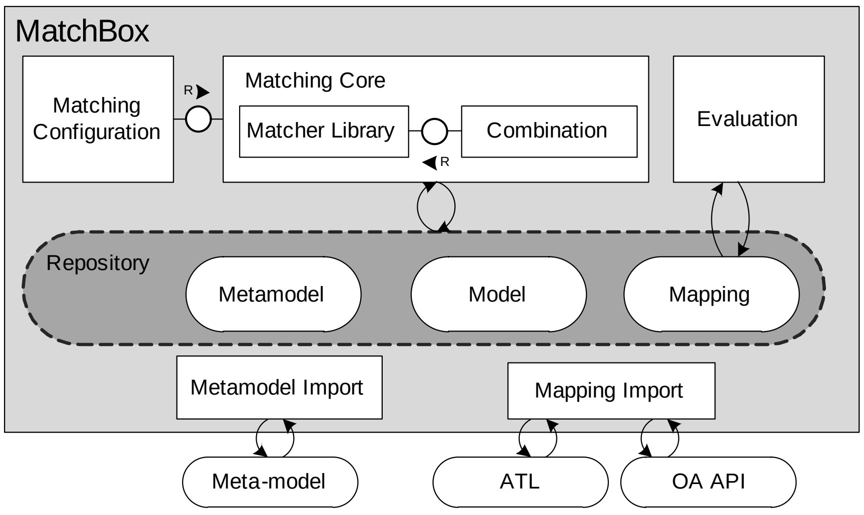


Figure B.1: Architecture of MatchBox

lowest value. *Average* evens out the matcher results, calculating the average. In order to satisfy a different importance of matcher results, *Weighted* computes the weighted sum of the results, according to user-defined weights.

- **Selection** The selection selects possible matches from the aggregated matrix according to a defined strategy. These are *Threshold*, *MaxN*, and *MaxDelta*. *Threshold* selects all matches exceeding a particular threshold. *MaxN* returns the N matches with the highest similarity values compared to all matches for a source and target element. Finally, *MaxDelta* chooses first, the match with the highest similarity value; second it returns those candidates within a certain relative range given by the value for delta. The range is defined as the interval $[Max - Max * Delta, Max]$. Furthermore, the selection strategies can be combined, such as *Threshold* and *MaxDelta*.

- **Direction** The direction is dedicated to a ranking of matched elements according to their similarity. In the *Forward* strategy elements of the larger metamodel (in total of elements) are selected for each element of the smaller schema. The *Backward* strategy is the inverse of the *Forward* strategy. Both strategies can be combined in the *Both* strategy, resulting in schema size independence. Consequently, a match has to have a similarity value for each direction forward and backward.

- **Combined similarity** Matchers which use other matchers have a need for means to combine the similarity values, e.g. like in the children or sibling matcher. This is covered by providing two strategies: *Average* and *Dice*. *Average* divides the sum of similarity values of all matches by the number of matches. The more optimistic approach, returning higher similarity values is *Dice*, it uses the dice coefficient [12]. It is computed by dividing the number of matchable elements by the count of all elements.

# Bibliography

[1] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the International Conference on Very Large Data Bases (VLDB'94)*, 1994.

[2] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Addison Wesley, 2006.

[3] Lyudmil Aleksandrov, Hristo Djidjev, Hua Guo, and Anil Maheshwari. Partitioning planar graphs with costs and weights. *Journal of Experimental Algorithmics*, 11, 2007.

[4] Michael Altenhofen, Thomas Hettel, and Stefan Kusterer. Ocl support in an industrial environment. In *6th OCL Workshop at the UML/MoDELS*, 2006.

[5] Cecilia R. Aragon, Catherine Schevon, Lyle A. McGeoch, and David S. Johnson. Optimization by simulated annealing: An experimental evaluation; part i, graph partitioning. *Operations Research*, 37:865–892, 1989.

[6] Uwe Aßmann, Steffen Zschaler, and Gerd Wagner. *Ontologies for Software Engineering and Software Technology*, chapter Ontologies, Metamodels, and the Model-Driven Paradigm, pages 249–273. 2006.

[7] Philip A. Bernstein, Alon Y. Levy, and Rachel A. Pottinger. A vision for management of complex models. *SIGMOD Record*, 29:55–63, 2000.

[8] Philip A. Bernstein, Sergey Melnik, Michalis Petropoulos, and Christoph Quix. Industrial-strength schema matching. *SIGMOD Record*, 33:38–43, 2004.

[9] Carsten Bönnen and Mario Herger. *SAP NetWeaver Visual Composer*. SAP Press, 2007.

[10] Cédric Brun and Alfonso Pierantonio. Model differences in the eclipse modeling framework. *Upgrade, Special Issue on Model-Driven Software Development IX*, pages 29–34, 2008.

[11] Jiazhen Cai, Xiaofeng Han, and Robert Endre Tarjan. n O(m log n)-time algorithm for the maximal planar subgraph problem. *SIAM Journal on Scientific Computing*, 22:1142–1162, 1993.

[12] Silvana Castano, Valeria De Antonellis, Maria G. Fugini, and Barbara Pernici. Conceptual schema analysis: techniques and applications. *ACM Transaction on Database Systems*, 23(3):286–333, 1998.

[13] Dirk G. Cattrysse and Luk N. Van Wassenhove. A survey of algorithms for the generalized assignment problem. *European Journal of Operational Research*, 60:260–272, 1992.

[14] Deepayan Chakrabarti and Christos Faloutsos. Graph mining: Laws, generators, and algorithms. *ACM Computer Survey*, 38:2, 2006.

[15] Yves Chiricota, Fabien Jourdan, and Guy Melançon. Software components capture using graph clustering. In *Proceedings of the 11th IEEE International Workshop on Program Comprehension (IWPC'03)*, 2003.

[16] Luigi P. Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. A (sub) graph isomorphism algorithm for matching large graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1367–1372, 2004.

[17] Isabel F. Cruz, Flavio Palandri Antonelli, Cosmin Stroe, Ulas C. Keles, and Angela Maduko. Using agreementmaker to align ontologies for OAEI 2010. In *Proceedings of the 5th International Workshop on Ontology Matching (OM'10)*, 2010.

[18] Jose de Sousa, Denivaldo Lopes, Daniela Barreiro Claro, and Zair Abdelouahab. A step forward in semi-automatic metamodel matching: Algorithms and tool. In *Proceedings of the 11th International Conference of Enterprise Information Systems (ICEIS'09)*, 2009.

[19] Peter J. Dickinson, Horst Bunke, Arek Dadej, and Miro Kraetzl. On graphs with unique node labels. *Graph-Based Representations in Pattern Recognition*, pages 13–23, 2003.

[20] Reinhard Diestel. *Graph Theory*. Graduate Texts in Mathematics. Springer, 2005.

[21] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1:269–271, 1959.

[22] Hristo Djidjev. A scalable multilevel algorithm for graph clustering and community structure detection. *Algorithms and Models for the Web-Graph*, pages 117–128, 2007.

[23] Hong Hai Do. *Schema Matching and Mapping-based Data Integration*. PhD thesis, University of Leipzig, 2006.

[24] Hong-Hai Do and Erhard Rahm. COMA – a system for flexible combination of schema matching approaches. In *Proceedings of the International Conference on Very Large Data Bases (VLDB'02)*, pages 610–621, 2002.

[25] Hong-Hai Do and Erhard Rahm. Matching large schemas: Approaches and evaluation. *Information Systems*, 32(6):857–885, 2006.

[26] Stijn Van Dongen. A cluster algorithm for graphs. Technical report, CWI: Centre for Mathematics and Computer Science, 2000.

[27] Prashant Doshi, Ravikanth Kolli, and Christopher Thomas. Inexact matching of ontology graphs using expectation-maximization. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7:90–106, 2009.

[28] Christian Drumm. *Improving schema mapping by exploiting domain knowledge*. PhD thesis, University of Karlsruhe, 2008.

[29] Christian Drumm, Matthias Schmitt, Hong-Hai Do, and Erhard Rahm. Quickmig: automatic schema matching for data migration projects. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management (CIKM '07)*, 2007.

[30] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. Formal integration of inheritance with typed attributed graph transformation for efficient VL definition and model manipulation. *IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 71–78, 2005.

[31] Leonhard Euler. Solutio problematis ad geometriam situs pertinentis. *Comment. Acad. Sci. U. Petrop.*, 8:128–140, 1736. Reprinted in Opera Omnia Series Prima, Vol. 7. pp. 1–10, 1766.

[32] J. Euzenat, A. Ferrara, L. Hollink, A. Isaac, C. Joslyn, V. Malaisé, C. Meilicke, A. Nikolov, J. Pane, M. Sabou, et al. Results of the ontology alignment evaluation initiative 2009. In *Proceedings of the 4th International Workshop on Ontology Matching (OM'09)*, 2009.

[33] Jérôme Euzenat and Pavel Shvaiko. *Ontology Matching*. Springer, 2007.

[34] Jérôme Euzenat and Petko Valtchev. Similarity-based ontology alignment in OWL-Lite. In *Proceedings of European Conference on Artificial Intelligence (ECAI'04)*, 2004.

[35] Joerg Evermann. Theories of meaning in schema matching: An exploratory study. *Information Systems*, 34:28–44, 2009.

[36] Marcos Didonet Del Fabro, Jean Bézivin, and Patrick Valduriez. Weaving models with the eclipse AMW plugin. In *Eclipse Summit Europe 2006*, 2006.

[37] Marcos Didonet Del Fabro and Patrick Valduriez. Semi-automatic model integration using matching transformations and weaving models. In *Proceedings of the 2007 ACM symposium on Applied computing (SAC '07)*, 2007.

[38] Jean-Rémy Falleri, Marianne Huchard, Mathieu Lafourcade, and Clémentine Nebut. Metamodel matching for automatic model transformation generation. In *Proceedings of the 11th international conference on Model Driven Engineering Languages and Systems (MoDELS '08)*, 2008.

[39] Christiane Fellbaum. *WordNet: An Electronic Lexical Database*. The MIT Press, May 1998. `http://wordnet.princeton.edu`.

[40] Gary William Flake, Robert Endre Tarjan, and Kostas Tsioutsiouliklis. Graph clustering and minimum cut trees. *Internet Mathematics*, 1(4):385–408, 2004.

[41] Pasi Fränti, Olli Virmajoki, and Ville Hautamäki. Fast pnn-based clustering using k-nearest neighbor graph. In *Proceedings of IEEE International Conference on Data Mining (ICDM'03)*, 2003.

[42] Avigdor Gal. Managing uncertainty in schema matching with top-k schema mappings. *Journal on Data Semantics*, 6:2006, 2006.

[43] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[44] Xinbo Gao, Bing Xiao, Dacheng Tao, and Xuelong Li. A survey of graph edit distance. *Pattern Analysis & Applications*, pages 1–17, 2010.

[45] Kelly Garcés, Frédéric Jouault, Pierre Cointe, and Jean Bézivin. Managing model adaptation by precise detection of metamodel changes. In *Proceedings of 5th European Conference on Model-Driven Architecture Foundations and Applications (ECMDA-FA09)*, 2009.

[46] K.J. Garcés-Pernett. *Une approche pour ladaptation et lévaluation de stratégies génériques dalignement de modèles*. PhD thesis, 2010.

[47] Michael Randolph Garey and David Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. 1979.

[48] Michelle Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99:7821–7826, 2002.

[49] Steve Gregrory. Local betweenness for finding communities in networks. Technical report, University of Bristol, 2008.

[50] Anika Gross, Michael Hartung, Toralf Kirsten, and Erhard Rahm. On matching large life science ontologies in parallel. In *Proceedings of Data Integration in the Life Sciences (DILFS'10)*, 2010.

[51] Giancarlo Guizzardi. On ontology, ontologies, conceptualizations, modeling languages, and (meta)models. *Frontiers in Artificial Intelligence and Applications, Databases and Information Systems IV*, pages 18–39, 2007.

[52] Magnus Halldorsson and Jaikumar Radhakrishnan. Greed is good: approximating independent sets in sparse and bounded-degree graphs. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, 1994.

[53] Seddiqui Hanif and Masaki Aono. Anchor-Flood: Results for OAEI 2009. In *Proceedings of Workshop on Ontology Matching (OM'09)*, 2009.

[54] Seddiqui Hanif and Masaki Aono. An efficient and scalable algorithm for segmented alignment of ontologies of arbitrary size. *Journal of Web Semantics*, 7:344–356, 2009.

[55] Erez Hartuv and Ron Shamir. A clustering algorithm based on graph connectivity. *Information processing letters*, 76(4-6):175–181, 2000.

[56] Jonathan Hayes and Claudio Gutiérrez. Bipartite graphs as intermediate model for RDF. *The Semantic Web – ISWC'04*, pages 47–61, 2004.

[57] Florian Heidenreich, Jendrik Johannes, Sven Karol, Mirko Seifert, and Christian Wende. Derivation and refinement of textual syntax for models. In *Proceedings of 5th European Conference on Model-Driven Architecture Foundations and Applications (ECMDA-FA09)*, 2009.

[58] Thomas Heinze. Matching algorithms on planar/reducible graphs for meta-model matching. Technical University (TU) Dresden, 2009. Minor thesis.

[59] Thomas Heinze. Graph-based clustering for large-scale metamodel matching. Master's thesis, Technical University (TU) Dresden, 2010.

[60] Stefan Helming and Michael Göbel. *ZOPP Objectives-oriented Project Planning*. Deutsche Gesellschaft für Technische Zusammenarbeit (GTZ) GmbH, 1997.

[61] John E. Hopcroft and Robert Endre Tarjan. Efficient planarity testing. *Journal of the ACM*, 21(4), 1974.

[62] John E. Hopcroft and J. K. Wong. Linear time algorithm for isomorphism of planar graphs (preliminary report). In *STOC'74*, 1974.

[63] Wei Hu, Ningsheng Jian, Yuzhong Qu, and Yanbing Wang. GMO: A graph matching for ontologies. In *Proceeedings of the K-Cap 2005 Workshop on Integrating Ontologies*, 2005.

[64] Wei Hu, Yuanyuan Zhao, Dan Li, Gong Cheng, Honghan Wu, and Yuzhong Qu. Falcon-ao: Results for OAEI 2007. In *Proceedings of the 2nd International Workshop on Ontology Matching (OM'07)*, 2007.

[65] Jun Huan, Wei Wang, and Jan Prins. Efficient mining of frequent subgraphs in the presence of isomorphism. In *Proceedings of IEEE International Conference on Data Mining (ICDM'03)*, 2003.

[66] Akihiro Inokuchi, Takashi Washio, and Hiroshi Motoda. An apriori-based algorithm for mining frequent substructures from graph data. *Principles of Data Mining and Knowledge Discovery*, pages 13—23, 2000.

[67] Petko Ivanov. Data model enhancement of a matching system based on comparative analysis of related tools. Master's thesis, Technical University (TU) Dresden, 2010.

[68] Yves R. Jean-Mary and Mansur R. Kabuka. ASMOV: Results for OAEI 2010. In *Proceedings of the 5th International Workshop on Ontology Matching (OM'10)*, 2010.

[69] Ningsheng Jian, Wei Hu, Gong Cheng, and Yuzhong Qu. Falcon-ao: Aligning ontologies with falcon. In *K-Cap 2005 Workshop on Integrating Ontologies*, 2005.

[70] Frédéric Jouault, Freddy Allilaire, Jean Bézivin, and Ivan Kurtev. ATL: A model transformation tool. *Scientific Computer Programs*, 72:31–39, 2008.

[71] Michael Jünger and Petra Mutzel. Maximum planar subgraphs and nice embeddings: Practical layout tools. *Algorithmica*, 16:33–59, 1996.

[72] Gerti Kappel, Elisabeth Kapsammer, Horst Kargl, Gerhard Kramler, Thomas Reiter, Werner Retschitzegger, Wieland Schwinger, and Manuel Wimmer. On models and ontologies – a layered approach for model-based tool integration. In *Proceedings of Modellierung 2006*, 2006.

[73] Gerti Kappel, Horst Kargl, Gerhard Kramler, Andrea Schauerhuber, Martina Seidl, Michael Strommer, and Manuel Wimmer. Matching metamodels with semantic systems – an experience report. In *Proceedings of Workshops at BTW 2007*, 2007.

[74] George Karypis, Eui-Hong Han, and Vipin Kumar. Chameleon: Hierarchical clustering using dynamic modeling. *IEEE Computer*, 32:68–75, 1999.

[75] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20:359, 1999.

[76] Jean François Djoufak Kengue, Jérôme Euzenat, and Petko Valtchev. Ola in the OAEI 2007 evaluation contest. In *Proceedings of the International Workshop on Ontology Matching (OM'07)*, 2007.

[77] David Kensche, Christoph Quix, Xiang Li, and Yong Li. Geromesuite: a system for holistic generic model management. In *Proceedings of the International Conference on Very Large Data Bases (VLDB '07). Demonstration.*, 2007.

[78] B.W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49:291–307, 1970.

[79] Nikhil S. Ketkar, Lawrence B. Holder, and Diane J. Cook. Subdue: compression-based frequent pattern discovery in graph data. In *Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations*, 2005.

[80] Philip Klein, Satish Rao, Monika Rauch, and Sairam Subramanian. Faster shortest-path algorithms for planar graphs. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, 1994.

[81] Dimitrios S. Kolovos, Davide Di Ruscio, Alfonso Pierantonio, and Richard F. Paige. Different models for model matching: An analysis of approaches to support model differencing. In *Proceedings of 2009 ICSE Workshop on Comparison and Versioning of Software Models (CVSM'09)*, 2009.

[82] Hanna Köpcke and Erhard Rahm. Frameworks for entity matching: A comparison. *Data & Knowledge Engineering*, 69:197–210, 2010.

[83] Mandy Krimmel and Joachim Orb. *SAP NetWeaver Process Integration (2nd Edition)*. SAP Press, 2010.

[84] Joseph. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. In *Proceedings of the American Mathematical Society*, volume 7, pages 48–50, 1956.

[85] H.W. Kuhn. The Hungarian method for the assignment problem. *Naval research logistics quarterly*, 2:83–97, 1955.

[86] Jacek P Kukluk, Lawrence B Holder, and Diane J Cook. Algorithm and experiments in testing planar graphs for isomorphism. *Graph Algorithms and Applications*, 5:313–356, 2003.

[87] Michihiro Kuramochi and George Karypis. Frequent subgraph discovery. In *Proceedings of IEEE International Conference on Data Mining (ICDM'01)*, 2001.

[88] Michihiro Kuramochi and George Karypis. GREW – a scalable frequent subgraph discovery algorithm. In *Proceedings of IEEE International Conference on Data Mining (ICDM'04)*, 2004.

[89] Michihiro Kuramochi and George Karypis. Finding frequent patterns in a large sparse graph. *Data Mining and Knowledge Discovery*, 11:243–271, 2005.

[90] C. Kuratowski. Sur le probléme des courbes gauches en topologie. *Fundamental in Mathematics*, 15:271–283, 1930.

[91] Ivan Kurtev, Mehmet Aksit, and Jean Bézivin. A global perspective on technological spaces. Available at: `http://wwwhome.cs.utwente.nl/~kurtev/TechnologicalSpaces.doc`, 2002.

[92] Bach Thanh Le and Rose Dieng-Kuntz. A graph-based algorithm for alignment of owl ontologies. In *Proceedings of Web Intelligence*, 2007.

[93] Jong Kook Lee, Seung Jae Seung, Soo Dong Kim, Woo Hyun, and Dong Han Han. Component identification method with coupling and cohesion. In *Proceedings of the Asia-Pacific Software Engineering Conference*, 2001.

[94] Yoonkyong Lee, Mayssam Sayyadian, Anhai Doan, and Arnon Rosenthal. Tuning schema matching software using synthetic scenarios. In *Proceedings of the International conference on Very Large Data Bases (VLDB'05)*, 2005.

[95] Juanzi Li, Jie Tang, Yi Li, and Qiong Luo. RiMOM: A dynamic multi-strategy ontology alignment framework. *IEEE Transactions on Knowledge and Data Engineering*, pages 1218–1232, 2008.

[96] Richard J. Lipton and Robert E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, pages 177–189, 1979.

[97] Henrik Lochmann. *HybridMDSD: Multi-Domain Engineering with Model-Driven Software Development using Ontological Foundations*. Technical University (TU) Dresden, 2009.

[98] Denivaldo Lopes, Slimane Hammoudi, and Zair Abdelouahab. Schema matching in the context of model driven engineering: From theory to practice. In *Proceedings of the International Conference on Systems, Computing Sciences and Software Engineering (SCSS'06)*, 2006.

[99] Eugene M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. In *FOCS*, pages 42–49, 1980.

[100] Jayant Madhavan, Philip A. Bernstein, and Erhard Rahm. Generic schema matching with Cupid. In *The VLDB Journal*, pages 49–58, 2001.

[101] Ming Mao, Yefei Peng, and Michael Spring. A harmony based adaptive ontology mapping approach. In *Proceedings of the 2008 International Conference on Semantic Web & Web Services (SWWS'08)*, 2008.

[102] Marc Wörlein, Alexander Dreweke, Thorsten Meinl, Ingrid Fischer, and Michael Philippsen. Edgar: the embedding-based graph miner. In *Proceedings of the International Workshop on Mining and Learning with Graphs*, 2006.

[103] Marc Wörlein, Thorsten Meinl, Ingrid Fischer, and Michael Philippsen. A quantitative comparison of the subgraph miners MoFa, gSpan, FFSM, and gaston. In *Proceedings of the Conference on Knowledge Discovery in Databases (PKDD'05)*, 2005.

[104] Silvano Martello and Paolo Toth. An algorithm for the generalized assignment problem. In *Operational Research*, 1981.

[105] Silvano Martello and Paolo Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, 1990.

[106] Takashi Matsuda, Tadashi Horiuchi, Hiroshi Motoda, and Takashi Washio. Extension of graph-based induction for general graph structured data. In *Proceedings of the 4th Pacific-Asia Conference on Knowledge Discovery and Data Mining, Current Issues and New Applications*, 2000.

[107] Sergey Melnik, Hector Garcia-molina, and Erhard Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proceedings of the 18th International Conference on Data Engineering (ICDE'02)*, 2002.

[108] Sergey Melnik, Erhard Rahm, and Philip A. Bernstein. Rondo: A programming platform for generic model. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data (SIGMOD '03)*, 2003.

[109] Ed Merks, David Steinberg, Frank Budinsky, and Marcelo Paternostro. *Eclipse Modeling Framework*. Addison-Wesley, second edition, 2009.

[110] Bruno T. Messmer and Horst Bunke. Efficient error-tolerant subgraph isomorphism detection. In *Shape, Structure and Pattern Recognition*, pages 231–240, 1994.

[111] Gary L. Miller, Shang-Hua Teng, William Thurston, and Stephen A. Vavasis. Geometric separators for finite-element meshes. *SIAM Journal on Scientific Computing*, 19:386, 1998.

[112] Peter Mucha. Musterbasiertes abbilden von metamodellen. Master's thesis, Technical University (TU) Dresden, 2010.

[113] Michel Neuhaus and Horst Bunke. An error-tolerant approximate matching algorithm for attributed planar graphs and its application to fingerprint classification. In *Proceedings of the 10th International Workshop on Structural and Syntactic Pattern Recognition*, 2004.

[114] Mark E. J. Newman. Fast algorithm for detecting community structure in networks. *Physical Review E*, 69, 2004.

[115] Object Management Group (OMG). *Business Process Modeling Notation (BPMN) Specification*, February 2006. Final Adopted Specification, OMG document dtc/06-02-01, `http://www.omg.org/docs/dtc/06-02-01.pdf`.

[116] Object Management Group (OMG). *OMG Systems Modeling Language (OMG SysML) Specification, Version 1.0*, 2007. OMG document ptc/2007-02-03.

[117] Per olof Fjällström. Algorithms for graph partitioning: A survey. *Computer and Information Science*, 3(10):10, 1998.

[118] OMG. Model driven architecture guide, version 1.0.1. 2003. omg/03-06-01.

[119] OMG. *Unified Modeling Language: Superstructure, version 2.0*. Object Management Group, 2005. formal/05-07-04.

[120] OMG. *Meta Object Facility (MOF) 2.0 Core Specification*. Object Management Group, 2006. formal/06-01-01.

[121] OMG. *Unified Modeling Language: Infrastructure, version 2.0*. Object Management Group, 2006. formal/05-07-05.

[122] Guillermo Peris and Andrés Marzal. Fast cyclic edit distance computation with weighted edit costs in classification. In *Proceedings of the International Conference on Pattern Recognition (ICPR'02)*, 2002.

[123] Eric Peukert, Henrike Berthold, and Erhard Rahm. Rewrite-techniques for performance optimzation of schema matching processes. In *Proceedings of 13th International Conference on Extending Database Technology (EDBT'10)*, 2010.

[124] Eric Peukert, Sabine Massmann, and Kathleen Koenig. Comparing similarity combination methods for schema matching. In Gesellschaft fr Informatik (GI), editor, *Proceedings of the GI Jahrestagung (1)*. Gesellschaft fr Informatik (GI), 2010.

[125] Erhard Rahm. Towards large-scale schema and ontology matching. *Schema Matching and Mapping*, pages 3–27, 2010.

[126] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350, 2001.

[127] R. Read and D. Corneil. The graph isomorphism disease. *Journal of Graph Theory*, 1:339–363, 1977.

[128] Kaspar Riesen, Michel Neuhaus, and Horst Bunke. Bipartite graph matching for computing the edit distance of graphs. *Graph-Based Representations in Pattern Recognition*, 4538:1, 2007.

[129] C. J. Van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, 1979.

[130] Dominique Ritze, Christian Meilicke, Ondrej Sváb-Zamazal, and Heiner Stuckenschmidt. A pattern-based ontology matching approach for detecting complex correspondences. In *Proceedings of Workshop on Ontology Matching (OM'09)*, 2009.

[131] Neil Robertson and P.D. Seymour. Graph minors. xx. wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92:325–357, 2004.

[132] Barna Saha, Ioana Stanoi, and Kenneth L. Clarkson. Schema covering: a step towards enabling reuse in information integration. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE'10)*, 2010.

[133] Alberto Sanfeliu and King-Sun Fu. Distance measure between attributed relational graphs for pattern recognition. *IEEE Transactions on Systems Science and Cybernetics*, 13:353–362, 1983.

[134] SAP AG. Integration scenario pos, 2009. `https://wiki.sdn.sap.com/wiki/display/CK/Integration+Scenario+POS`. Last visited June 2011.

[135] SAP AG. Process composer, 2011. `http://sap.com/platform/netweaver/components/sapnetweaverbpm/index.epx`. Last visited June 2011.

[136] August-Wilhelm Scheer and Frank Habermann. Enterprise resource planning: making erp a success. *Communications of the ACM*, 43:57–61, April 2000.

[137] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905, 2000.

[138] Pavel Shvaiko and Jérôme Euzenat. A survey of schema-based matching approaches. *Journal on Data Semantics*, 4:146–171, 2005.

[139] Horst D. Simon and Shang hua Teng. How good is recursive bisection? *SIAM Journal on Scientific Computing*, 18:1436–1445, 1997.

[140] Marko Smiljanic, Maurice van Keulen, and Willem Jonker. Using element clustering to increase the efficiency of xml schema matching. In *Proceedings of the ICDE Workshops*, page 45, 2006.

[141] Daniel A. Spielman and Shang hua Teng. Spectral partitioning works: Planar graphs and finite element meshes. *Linear Algebra and its Applications*, 421:284–305, 2007.

[142] Dave Steinberg, Frank Budinski, Marcelo Paternostro, and Ed Merks. *EMF Eclipse Modelling Framework*. Addison Wesley, 2 edition, 2008.

[143] Gunther Stuhec. Using CCTS modeler warp 10 to customize business information interfaces. *SAP Developer Network*, 2007.

[144] Jeffrey D. Ullmann. An algorithm for subgraph isomorphism. *Journal of the ACM*, 23(1):31–42, 1976.

[145] Gabriel Valiente. Graph isomorphism and related problems. Technical report, Technical University of Catalonia, 2001.

[146] Konrad Voigt. Generation of language-specific transformation rules based on metamodels. In *Proceedings of the 1st IoS PhD Symposium 2008 at I-ESA'08*, 2008.

[147] Konrad Voigt. Towards combining model matchers for transformation development. In *Proceedings of 1st International Workshop on Future Trends of Model-Driven Development at ICEIS'09 (FTMDD'09)*, 2009.

[148] Konrad Voigt. Semi-automatic matching of heterogeneous model-based specifications. In *Proceedings of the Software Engineering 2010 (Doc. Symp.) (SE'10)*, 2010.

[149] Konrad Voigt. Evaluation data series, 2011. `http://www.voggy.de/files/evaluation.zip`.

[150] Konrad Voigt and Thomas Heinze. Meta-model matching based on planar graph edit distance. In *Proceedings of International Conference on Model Transformation 2010 (ICMT'10)*, 2010.

[151] Konrad Voigt, Petko Ivanov, and Andreas Rummler. MatchBox: Combined meta-model matching for semi-automatic mapping generation. In *Proceedings of the 2010 ACM Symposium on Applied Computing (SAC'10)*, 2010.

[152] Klaus Wagner. Über eine erweiterung eines satzes von kuratowski. *Deutsche Mathematik*, 2:280–285, 1937.

[153] Robert A. Wagner and Michael J. Fischer. The string to string correction problem. *Journal of the ACM*, 21(21):168–173, 1974.

[154] Peng Wang and Baowen Xu. Lily: Ontology alignment results for oaei 2009. In *Proceedings of the 5th International Workshop on Ontology Matching (OM'09)*, 2009.

[155] Fang Wu and Bernardo A. Huberman. Finding communities in linear time: a physics approach. *The European Physical Journal B-Condensed Matter and Complex Systems*, 38(2):331–338, 2004.

[156] Wei Wu, Yuzhong Qu, and Gong Cheng. Matching large ontologies: A divide-and-conquer approach. *Data Knowledge and Engineering*, 67:140–160, October 2008.

[157] Xifeng Yan and Jiawei Han. gSpan: graph-based substructure pattern mining. In *Proceedings of the IEEE International Conference on Data Mining (ICDM'02)*, 2002.

[158] Xifeng Yan and Jiawei Han. CloseGraph: mining closed frequent graph patterns. In *Proceedings of the ninth ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD'03)*, 2003.

[159] Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal of Computing*, 18:1245–1262, 1989.

[160] Xiao Zhang, Qian Zhong, Juanzi Li, and Jie Tang. RiMOM results for oaei 2010. In *Proceedings of the 5th International Workshop on Ontology Matching (OM'10)*, 2010.

[161] Zhi Zhang, Pengfei Shi, Haoyang Che, and Jun Gu. An algebraic framework for schema matching. *Informatica*, 19(3):421–446, 2008.

[162] Zhongping Zhang, Rong Li, Shunliang Cao, and Yangyong Zhu. Similarity metric for xml documents. In *Proceedings of Workshop on Knowledge and Experience Management*, 2003.

# List of Figures

2.1 MOF three layer architecture and example . . . . . . . . . . . 10
2.2 Generic matcher receiving two elements as input . . . . . . . 11
2.3 Architecture and process of a generic matching system . . . . 12
2.4 Classification of matching techniques . . . . . . . . . . . . . . 15
2.5 Examples for global, local, and region-based matching . . . . 17
2.6 Examples for a graph . . . . . . . . . . . . . . . . . . . . . . . 20
2.7 Package, class, attribute, and operation mapping . . . . . . . 21
2.8 Inheritance-, reference-based, and complete metamodel graph 22
2.9 Flattened containment and reference metamodel tree . . . . . 23
2.10 Example of a graph, a minor, and edge contraction . . . . . . 25
2.11 Example of a graph and planarity . . . . . . . . . . . . . . . . 26
2.12 The non-planar graphs $K_5$ and $K_{3,3}$ . . . . . . . . . . . . . . . 26
2.13 Subset relation between general, planar, and trees . . . . . . . 27
2.14 Classification of graph matching algorithms . . . . . . . . . . 29
2.15 Example of a graph, a pattern and embeddings of this pattern 31
2.16 Classification of graph mining algorithms . . . . . . . . . . . . 32
2.17 Classification of graph partitioning and clustering algorithms . 33

3.1 Example of data integration for POS and ERP . . . . . . . . . 36
3.2 Details of retail store data integration example . . . . . . . . 37
3.3 Metamodel example POS and ERP . . . . . . . . . . . . . . . 38
3.4 Problem hierarchy . . . . . . . . . . . . . . . . . . . . . . . . 41
3.5 Objective hierarchy . . . . . . . . . . . . . . . . . . . . . . . . 44
3.6 Steps of our problem solving approach . . . . . . . . . . . . . 47

5.1 Example of source and target graphs, and a common subgraph 64
5.2 Example metamodel excerpts and corresponding graphs . . . 70
5.3 Example calculation of planar graph edit distance . . . . . . . 71
5.4 Illustrative example of $k$-max degree with user interaction . . 73
5.5 General mining based matching process . . . . . . . . . . . . 74
5.6 Example for mining graph model types using similarity classes 78
5.7 Design pattern matcher process . . . . . . . . . . . . . . . . . 79
5.8 Example for pattern mining by the design pattern matcher . . 83

5.9 Redundancy matcher process . . . . . . . . . . . . . . . . . . 85
5.10 Example for pattern mining by the redundancy matcher . . . 88

6.1 Processing steps of partition-based matching and assignment . 92
6.2 Example metamodel and the corresponding graph . . . . . . . 98
6.3 Example calculation for planar partitioning . . . . . . . . . . 99
6.4 Example of the partition merging phase . . . . . . . . . . . . 102
6.5 Partition matching without assignment . . . . . . . . . . . . . 103
6.6 Partition matching with threshold-based assignment . . . . . 104
6.7 Partition matching with quantile-based assignment . . . . . . 106
6.8 Partition matching with Hungarian assignment . . . . . . . . 107
6.9 Partition matching with generalized assignment . . . . . . . . 108

7.1 Processing of our framework MatchBox . . . . . . . . . . . . . 116
7.2 Example of aggregation and selection of matcher results . . . 118
7.3 Size of ESR metamodels . . . . . . . . . . . . . . . . . . . . . 122
7.4 Source and target metamodel element ratio for ESR . . . . . . 123
7.5 Size of ESR gold-standard mappings . . . . . . . . . . . . . . 123
7.6 Coverage of ESR gold-standard mappings . . . . . . . . . . . 124
7.7 Name and token overlap for ESR metamodels . . . . . . . . . 125
7.8 Number of edges for ESR metamodels . . . . . . . . . . . . . 125
7.9 Maximal degree for ESR metamodels . . . . . . . . . . . . . . 126
7.10 Loss in edges for ESR metamodel graphs . . . . . . . . . . . . 126
7.11 Size of ATL metamodels . . . . . . . . . . . . . . . . . . . . . 127
7.12 Source and target metamodel element ratio for ATL . . . . . . 128
7.13 Size of ATL gold-standard mappings . . . . . . . . . . . . . . 128
7.14 Coverage of ATL gold-standard mappings . . . . . . . . . . . . 129
7.15 Name and token overlap for ATL metamodels . . . . . . . . . 130
7.16 Number of edges for ATL metamodels . . . . . . . . . . . . . 130
7.17 Maximal degree for ATL metamodels . . . . . . . . . . . . . . 130
7.18 Loss in edges for ATL metamodel graphs . . . . . . . . . . . . 131
7.19 Baseline quality results for the ESR and ATL data sets . . . . . 134
7.20 Result quality for the ESR data using the GED . . . . . . . . . 136
7.21 Result quality for the ATL data using the GED . . . . . . . . . 137
7.22 Delta of quality for increasing k used only by the GED . . . . 138
7.23 Delta of quality for increasing k added to the result . . . . . . 138
7.24 Distribution of precision delta for GED . . . . . . . . . . . . 139
7.25 Examples for quality changes by the GED matcher . . . . . . . 140
7.26 Result quality for mining-based matchers . . . . . . . . . . . . 143
7.27 Quality and runtime baseline for partition-based matching . . 145
7.28 Comparison of precision and recall dependent on size . . . . . 146
7.29 Comparison of memory and runtime dependent on size . . . . 147
7.30 Quality and runtime for threshold-based assignment . . . . . 148
7.31 Quality and runtime for quantile-based assignment . . . . . . 148

7.32 Quality and runtime for hungarian assignment . . . . . . . . 149
7.33 Quality and runtime for generalized assignment . . . . . . . . 150
7.34 Summary for partition based matching . . . . . . . . . . . . 151
7.35 Summary for flat partition based matching . . . . . . . . . . . 152

A.1 Example of an import from ATL . . . . . . . . . . . . . . . . . . 167
A.2 Excerpt of the ATL metamodel . . . . . . . . . . . . . . . . . . . 169
A.3 Excerpt of the MatchBox model . . . . . . . . . . . . . . . . . . 169

B.1 Architecture of MatchBox . . . . . . . . . . . . . . . . . . . . . 176

# List of Tables

3.1 Overview of requirements . . . . . . . . . . . . . . . . . . . . . 46

4.1 Overview on related structural matching approaches . . . . . 58
4.2 Overview on related large-scale matching systems . . . . . . . 60

5.1 Overview and comparison of graph matching algorithms . . . 66
5.2 Overview and comparison of graph mining algorithms . . . . 77
5.3 Contributions of structural graph-based matchers . . . . . . . 90

6.1 Graph clustering and partitioning algorithms . . . . . . . . . . 95
6.2 Comparison of the four assignment approaches . . . . . . . . 110
6.3 Contributions of graph-based partitioning and assignment . . 111

7.1 Comparison of metrics for ESR and ATL data set . . . . . . . . 132
7.2 Results and limitations of our concepts . . . . . . . . . . . . . 153

A.1 Mapping from ATL to the MatchBox mapping metamodel . . . 171

# List of Algorithms

5.1 Planar graph edit distance algorithm . . . . . . . . . . . . . . 68
5.2 Design pattern matcher algorithm . . . . . . . . . . . . . . . . . 80
5.3 Pattern mining in design pattern matcher (minePatterns) . . . 81
5.4 Redundancy matcher algorithm . . . . . . . . . . . . . . . . . . 85
5.5 Pattern identification for redundancy matcher (minePatterns) 86
6.1 Planar graph-based partitioning algorithm outline . . . . . . . 97
A.1 ATL-Rule Import . . . . . . . . . . . . . . . . . . . . . . . . . 170
A.2 Binding Import (importB) . . . . . . . . . . . . . . . . . . . . . 172